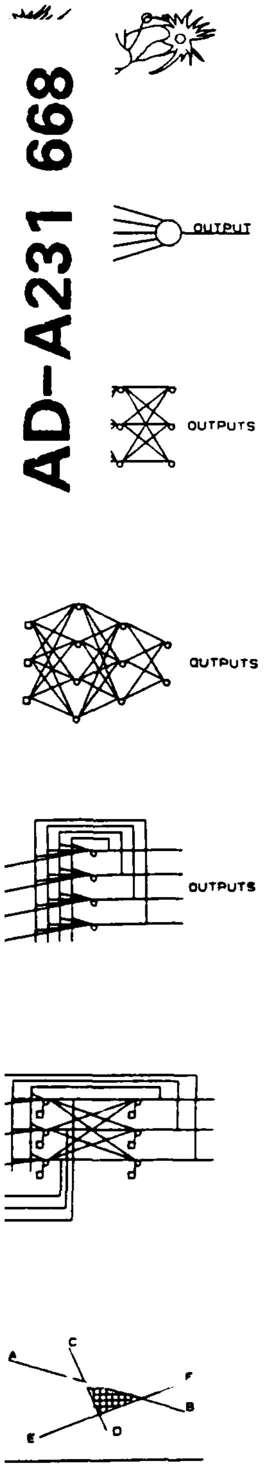
Army Corps
Engineers

AD-A231 668



TECHNICAL REPORT O-90-1

2

ANALYSIS OF WIDEBAND BEAMFORMERS DESIGNED WITH ARTIFICIAL NEURAL NETWORKS

by

Cary Cox

Instrumentation Services Division

DEPARTMENT OF THE ARMY
Waterways Experiment Station, Corps of Engineers
3909 Halls Ferry Road, Vicksburg, Mississippi 39180-6199

December 1990

Final Report

Prepared for DEPARTMENT OF THE ARMY
Assistant Secretary of the Army (R&D)
Washington, DC 20315

Under Work No. A91D-LR-006

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1990	3. REPORT TYPE AND DATES COVERED Final report

4. TITLE AND SUBTITLE Analysis of Wideband Beamformers Designed with Artificial Neural Networks	5. FUNDING NUMBERS
6. AUTHOR(S) Cary Cox	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) USAE Waterways Experiment Station Instrumentation Services Division 3909 Halls Ferry Road Vicksburg, MS 39180-6199	8. PERFORMING ORGANIZATION REPORT NUMBER Technical Report O-90-1
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Assistant Secretary of the Army Washington, DC 20315	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The ability to determine the direction of an approaching wave is demonstrated by using an artificial neural network and a beamformer array. To demonstrate the ability of a neural network to learn a satisfactory solution to this problem, simulations were performed to test the system's sensitivity to several variables. These variables include amplitude range, noise level, frequency bandwidth, linear and nonlinear inputs, the number of sensor inputs, and the number of hidden units used in the network. Simulations are provided for both narrowband and wideband signals. An empirical test and a comparison between ANN beamformers and FFT beamformers are also used to demonstrate the strengths and weaknesses of the system. A design example and a description of the simulation program are also included. A brief tutorial on beamformers and neural networks is also provided.

14. SUBJECT TERMS Artificial neural networks Feedforward networks Backpropagation Direction of arrival Beamformers	15. NUMBER OF PAGES 244
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

PREFACE

This investigation was performed by personnel of the US Army Engineer Waterways Experiment Station (WES) under the In-House Laboratory Independent Research Program (ILIR). The work was performed under Work Number A91D-LR-006, "Analysis of Wideband Beamformers Designed with Artificial Neural Networks".

The study was conducted under the general supervision of Messrs. George P. Bonner, Chief, Instrumentation Services Division and Leiland M. Duke, Chief, Operations Branch. This report was written by Dr. Cary B. Cox, Data Reduction and Digital Support Section, Instrumentation Services Division. This study was also published as a dissertation for Mississippi State University under the direction of Dr. F. M. Ingels of the Electrical Engineering Department.

The Commander and Director of WES during preparation of this report was COL Larry B. Fulton. Technical Director was Dr. Robert W. Whalin.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Artificial Neural Networks

Artificial Neural Networks (ANN) is a term used to describe single or multilevel networks that use an artificial neuron as the main processing element. These artificial neurons are designed to emulate the functions performed by the biological neurons found in the cerebral cortex. It is obvious to most computer scientists that there are many classes of problems that are extremely difficult to solve on a standard sequential von Neumann computer but are very easy for the mammalian brain to process. Pattern recognition is one example of these problems. Programming a von Neumann computer to recognize objects or patterns is very difficult and minor changes in the object being recognized can easily cause the programs to fail. The developing mind of a 2 year old child however can easily recognize faces of parents, siblings, or hundreds of objects. These recognitions can easily be made when distortions or modifications such as changes in scale, rotation, or translation occur. Even lower forms of animal life such as dogs and cats can recognize sounds, objects, and smells.

In recent years an effort has been made to solve such problems by emulating the functions of the cerebral cortex. The cerebral cortex is composed of around 10 billion to 100 billion cells known as neurons. [14] These neurons are connected in hierarchical networks with over 10,000 billion to 100,000 billion connections. A diagram of a biological neuron is shown in Figure 1 [9]. There are four major parts of the neuron.

(1) The "Soma" which is the main body of the neuron cell.

(2) An "Axon" which is attached to the soma and produces an electrical pulse to be transmitted to other neurons.

(3) The "Dendrites" which receive these electrical signals.

(4) The "Synapses" which form the connections where two dendrites meet.

In order to emulate these cells, mathematical models of biological neurons have been devised. The sophistication of these models ranges from very simple to extremely complex. Detailed mathematical models of the biological neuron can be very complex. Some researchers believe that these detailed models should be used, while others believe that only the very basic functions of the neuron should be included in a model. This controversy can be partially explained by the uncertainty that many neurologists have as to exactly which functions of

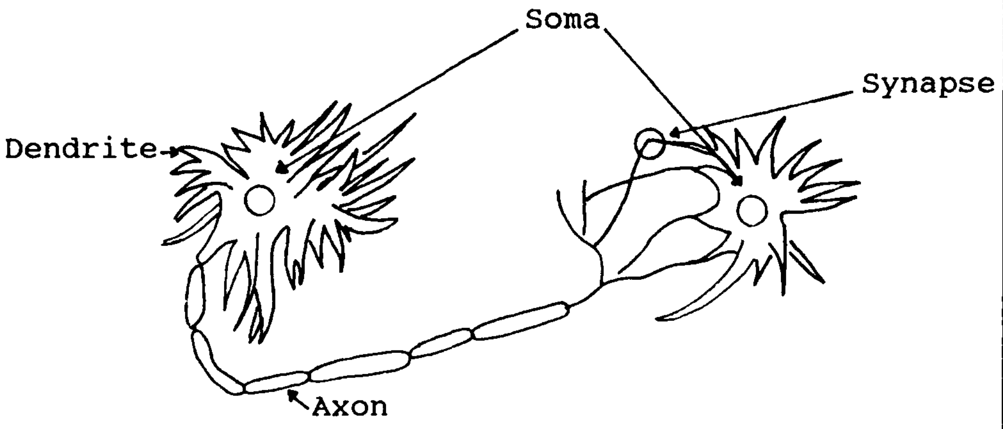


Figure 1

Biological Neurons

the neuron are used to process information and which are purely biological functions. There are however several very basic models that are used in many simulations, these consist of a special function operating on the summation of N weighted inputs. [4]

$$NET_j = \sum_{i=1}^{N} W_i X_i \qquad (1.1)$$

$$OUTPUT_j = f(NET_j) \qquad (1.2)$$

Where X_i are the inputs from sensors or previous neurons.

f is a special function.

W_i are the weights connecting neuron i to neuron j.

Many different methods of connecting and training neurons have been devised. The function of the electrical connections from the output of one artificial neuron to the input of another artificial neuron (X_i) is analogous to the function of the synapses that connect biological neurons to one another. The strength of the connection that the synapse has with the neuron influences the ability of the neuron to respond. The strength of these synapses is analogous to the weights (W_i) in the numerical model and the values assigned to these weights determine the function the artificial neuron will perform.

One of the important characteristics of ANNs is that they can be trained instead of programmed. Training can be accomplished in a variety of ways. Weights can be preset based on a numerical analysis of the problem and the network's configuration or they can be taught acceptable values. This teaching process is an iterative process performed by presenting the inputs to the network with a set of patterns to be recognized and adjusting weights in order to provide the desired output response. This later method is of great importance in neural network design since a formal mathematical analysis of many problems can be extremely difficult.

An impressive amount of research which utilizes ANNs is currently being conducted. There is an expectation that integrated circuits can be designed which implement ANNs using large scale integration. By designing dense ANN circuits a large number of problems such as pattern recognition, verbal word distinction, robotic controls, and signal processing may be solved.

1.2 Beamformer Arrays

A beamformer array is an array of sensors that is used to process signals in which direction is an important variable. In some applications, in which the direction of the source of a signal is known, a beamformer array can be steered to pickup the desired signal and filter out the noise being transmitted from other directions. Another important application is one in which the direction from which the information being transmitted is not known. In this case the direction can be determined by analyzing the data received by the array. Several different methods of determining the direction of a wavefront can be employed. If the signal being transmitted has a narrow bandwidth, analog or digital techniques can be used to analyze the array's output. From this analysis the direction of the wavefront can be determined. However when the signal being transmitted is composed of a wideband of frequencies more complex methods must be employed to filter the information in time in order to process only narrow bands of frequencies. The processing of this information from wideband beamformers can require the use of many filters and operational amplifiers or a substantial digital signal processing effort in order to discern the direction of the wave.

1.3 Purpose of Dissertation

The purpose of this dissertation is to design and analyze methods of determining the direction of arrival of a wideband waveform using a beamformer array and an artificial neural network. Recent research with neural networks has demonstrated their ability to distinguish between different patterns. These patterns consist of events such as sonar signals [10], alphabet characters [11], and radar signals [12]. Many of these efforts use preprocessing such as frequency analysis or filtering to extract important features from the data that would be difficult for the neural network to process. This preprocessing is performed on the input signal prior to insertion into the neural network. In this dissertation outputs from arrays of sensors are used with little or no preprocessing to demonstrate how the phase information received from these arrays can be used to train ANNs to distinguish direction. Comparisons will be made to demonstrate ANN's sensitivity to the following variables.

Design variables:

(1) The number of hidden units.

(2) The number of sensor inputs.

(3) Preprocessing of inputs.

Input variables:

(4) The noise level.

(5) The amplitude range.

(6) The signal's bandwidth

Other variables:

(7) The network's dependency on time.

(8) The different training methods being used.

When trying to determine a suitable problem to be worked with ANNs as opposed to von Neumann computers, two important criteria should be considered.

(1) Does the problem present computational difficulties on a von Neumann computer?

(2) Is the problem one that the human brain can easily process?

The first criteria is definitely met for this problem. The filtering of the sensor's outputs into narrowband signals will require either analog circuits or analog to digital converters and a microprocessor or special digital signal processing hardware. If a digital solution is used it will require many multiplications and additions for each band of directions to be tested. The second criteria is also met. When viewing a smooth surface wave on a lake or ocean it is easy to visually determine the approximate direction from which the wave approaches. The human brain can process this information in realtime with very little effort when only one wave is present. However more effort is required to determine this direction if the surface is turbulent or if waves from multiple directions are present. Since both of these criteria are met the problem is assumed to be one in which ANNs could provide a quick and reasonably accurate result.

1.4 Outline of Dissertation

In this dissertation the second and third chapters contain a tutorial on ANNs and beamformer arrays. The information contained in them is a summary of applicable information concerning ANNs and Beamformers, such as might be found in a text or reference book such as [2], [3], [4], [5], [6], [16]. Chapter 2 discusses artificial neuron models, different types of neural networks, and the evolution of learning algorithms. Mathematical and graphic examples showing how ANNs are used to separate linearly separable patterns are also presented. In Chapter 3 methods of determining a wave's direction from a beamformer array are presented for both narrowband and wideband signals. The classical method of solving the problem is presented along with some simulation results. A discussion of previous work is also included. Chapter 4 provides a design and analysis of an ANN narrowband beamformer. The learning methods are discussed and a mathematical analysis is used to calculate the weights and expected output of the network. Plots for both learned weights and calculated weights are presented that demonstrate the networks ability to learn and its sensitivity to amplitude and noise. Chapter 5 provides a design and analysis of an ANN wideband beamformer. A mathematical analysis similar to the one provided for narrow band beamformers is presented in Chapter 5 for wideband beamformers. Chapter 6 presents a demonstration of ANN wideband beamformers on an empirical test using seismic data. First some of the acquired data is used to train a network. After the network is trained additional test signals are passed through the network to test its accuracy. In Chapter 6 a comparison between ANN beamformers and FFT beamformers is presented. This comparison is made for both narrowband and wideband beamformers. Chapter 7 presents some design criteria and instructions on how the simulation program can be used to help test a design's configuration. Chapter 8 presents a brief summary on the types of neural computing hardware that are available commercially at the time of this writing. Chapter 9 presents the conclusions and recommendations for further research.

The data for most plots contained in this dissertation were calculated using a program called "VARIABLE WIDEBAND BEAMFORMER NEURAL NETWORK" (VWBBFNN). This program was written specifically for this dissertation. A complete listing of the program is included in Appendix A along with instructions for its use and a sample input and output listing. This program can be used to train networks of 1, 2, 3, or 4 levels of ANNs. It can process wideband or narrowband signals. The amplitude, frequency, and noise ranges are adjustable. The number of inputs and the location of sensors and the network's configuration of hidden units can also be designated. After a network is trained simulated data is used to test the network. The output response of this simulated data with arrival angles of 0 to 90 degrees is recorded on a disk file for further analysis or plotting. The ideal response of a network will produce a 1 when the angle is within the desired band and a 0 elsewhere. A plot of an ideal response is shown in Figure 2.

It is desirable to design or train networks that are independent of both time and signal frequency so the network will work in realtime with wideband signals. It is also desirable for the network to be as insensitive to noise and amplitude changes as possible. When reviewing the outputs of these networks it is found that they can almost always be improved by performing some output averaging. Therefore several of the plots are presented in two forms. In the first form both the maximum and minimum values of the network over the period of one cycle are presented. In the second form the average values of the network are presented. These averages are found by evaluating the network at 20 random times and averaging the results. This improvement can easily be implemented by shunting the output of the final neuron with a capacitor.

The time required to train these networks can become quite lengthy. The simulations were performed on a variety of computers from PCs to a CRAY Y-MP. In Chapter 8 some of these times are documented to help explain the advantages that might be gained by using neural computer boards to train and simulate neural networks.

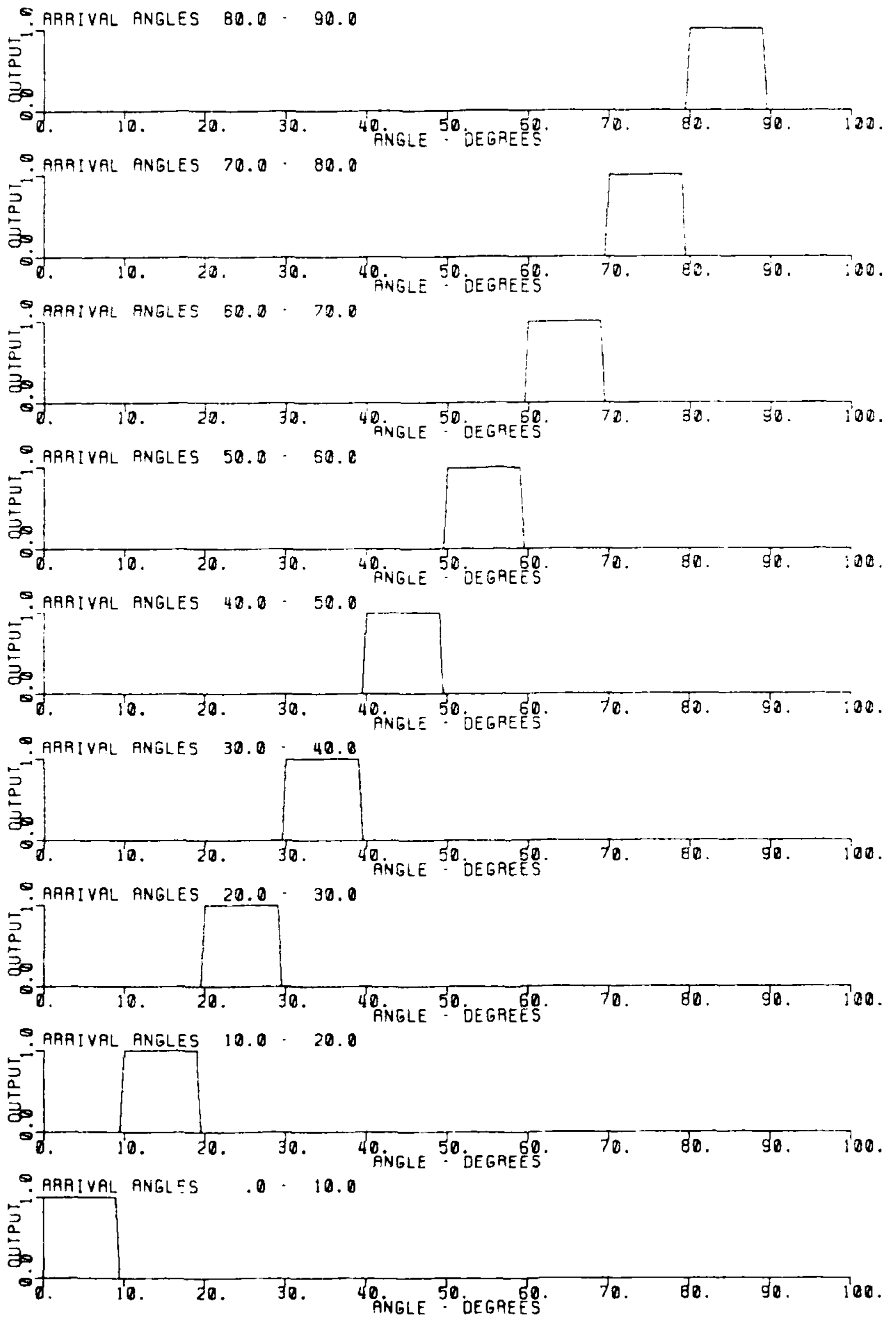


Figure 2

Ideal Response of an ANN Beamformer

Chapter 2

Tutorial On Artificial Neural Networks

2.1 Artificial Neuron Models

There are many different models of the basic neuron processing elements. Three of the more important and common ones are: the Adaptive Linear Element, the Perceptron, and the Backpropagation Perceptron. Each of these has played an important role in the history of artificial neural networks.

2.1.1 Adaptive Linear Element

The Adaptive Linear Element (ADALINE) or the Multiple Adaptive Linear Element (MADALINE) are an early form of artificial neurons developed by Widrow [7] in the early 1960's. ADALINEs primarily act as adaptive filters. They can be implemented by using an operational amplifier, a feedback resistor, and variable resistors connected to its inputs. An example is show in Figure 3.

These ADALINEs cannot be used to separate regions in a pattern recognition problem, since they give analog rather than discrete answers to a problem. ADALINEs are one of the oldest forms of artificial neurons.

They have been used in nulling radar jammers and adaptive equalizers in telephone lines. They can be modeled with the equation $I = \sum_{i=1}^{N} X_i W_i$, which is the same form used in a convolution filter. Therefore ADALINEs can be viewed as adaptive filters. When multiple ADALINEs are used to map a vector representation into another domain the result can be written as a matrix multiplication **V** = **WX**. It is important to note that multiple network levels of MADALINES can always be represented in only one level. For example a three level network **V** = $\mathbf{W}_1$**X**, **X** = $\mathbf{W}_2$**Y**, **Y** = $\mathbf{W}_3$**Z** can be represented as **V** = $\mathbf{W}_1\mathbf{W}_2\mathbf{W}_3$**Z** or **V** = **WZ** where **W** is the product of the three matrices, $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{W}_3$. Since the network can be represented in just one level, training is greatly simplified.

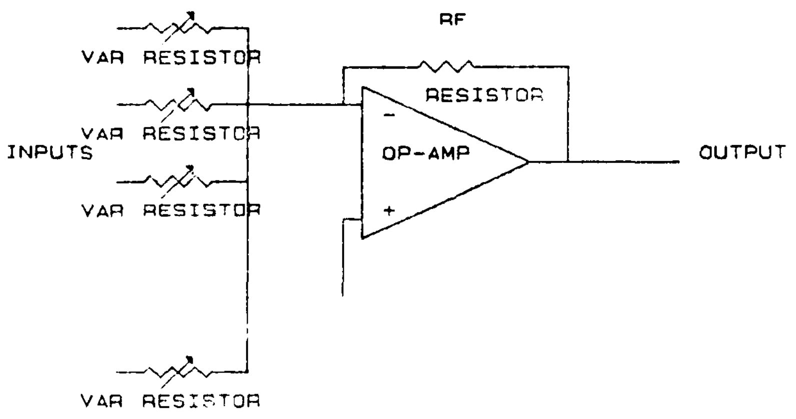


Figure 3

Diagram of ADALINE

2.1.2 Perceptrons

Perceptrons are also an early form of artificial neurons. They were designed by Rosenblat in 1956. They are also made from operational amplifiers and resistors, but a comparator is connected to the output of the amplifier in order to yield a binary 1 or 0 output when the signal is above or below a specified threshold value respectively. The threshold values can be implemented by using a constant voltage applied to one of the inputs. A diagram of a Perceptron is shown in Figure 4.

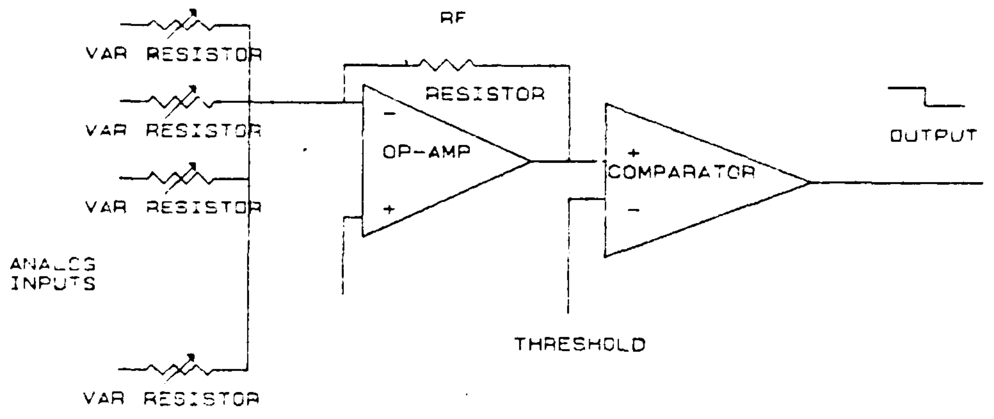


Figure 4

Model of Perceptron

2.1.3 Backpropagation Perceptrons

One commonly used model for the artificial neuron is the backpropagation perceptron. It can be implemented with an operational amplifier, a feedback resistor, N variable input resistors, and a nonlinear function module. When the nonlinear function is a simple comparator the model is equivalent to a Rosenblat perceptron. However this comparator is usually replaced by the function $f(X) = \frac{1}{1+e^{X}}$ to aid in the training of the network with the backpropagation algorithm. This model of the neuron is usually designated as shown in Figure 5 and is connected in hierarchical networks as shown is Figure 6.

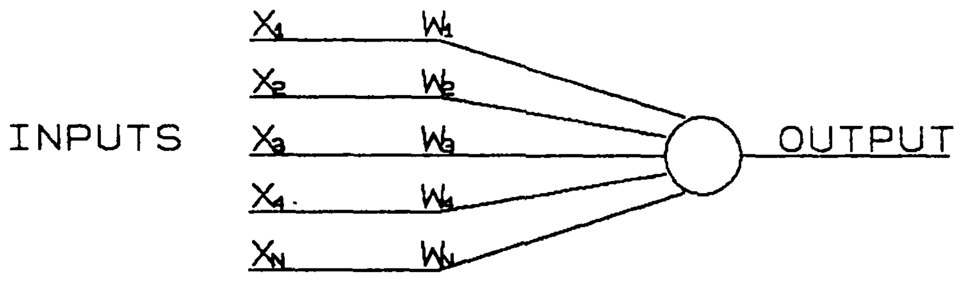


Figure 5

Backpropagation Perceptron

In addition to these inputs known as excitatory inputs many neurons contain an inhibitory input. When exerted the inhibitory input will cause the neuron to be turned off regardless of the excitatory inputs. These inhibitory inputs are sometimes used on all neurons in one level of a network, so that one and only one neuron in that level will turn on. These levels are know as "winner take all" levels. [4] This form of competition between neurons is very similar to the operation of biological networks and helps to develop contrast between results.

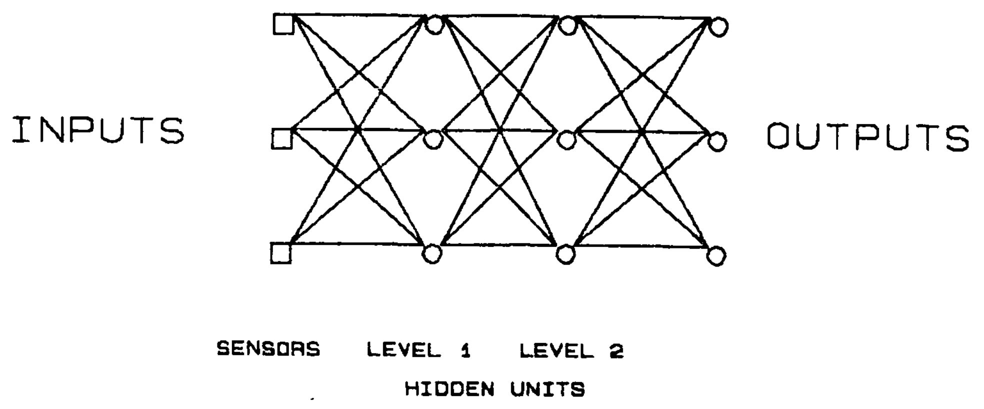


Figure 6

Multilevel ANN

These perceptrons are very similar to threshold elements [13] and can perform simple logical functions. The inputs to a perceptron can be either analog or digital. Its outputs however are digital. This means that the later levels of a neural network can be used to perform logical functions. Consider the following three examples.

CASE 1:

Design an "AND" function. (C = A .AND. B)

The equation for the perceptrons are:

$$AW(1)+BW(2)+W(3) > 0 \text{ for } A=1 \text{ and } B=1 \quad (2.1)$$
$$< 0 \text{ otherwise.}$$

Therefore

$$W(1)+W(2)+W(3) > 0 \quad (2.2)$$

$$W(1) \quad +W(3) < 0 \quad (2.3)$$

$$W(2)+W(3) < 0 \quad (2.4)$$

$$W(3) < 0. \quad (2.5)$$

If W(3) is set equal to -1 the inequalities become

$$W(1)+W(2) > 1 \quad (2.6)$$

$$W(1) \quad < 1 \quad (2.7)$$

$$W(2) < 1. \quad (2.8)$$

If W(1) and W(2) are chosen to be 0.7 the three inequalities will hold and the element will function as an "AND" gate.

$$.7A+.7B > 1.0 \quad (2.9)$$

Many weights other than 0.7 could also be used to solve the inequalities. Threshold values other than -1.0 could also be chosen. It is these weights and threshold values that must be set or learned by the network in order to perform the desired function.

CASE 2:

Design an "OR" function. (C = A .OR. B)

The equation for the Perceptrons are:

$$AW(1)+BW(2)+W(3) > 0 \tag{2.10}$$

$$AW(1) \qquad +W(3) > 0 \tag{2.11}$$

$$BW(2)+W(3) > 0 \tag{2.12}$$

$$W(3) < 0. \tag{2.13}$$

If W(3) is set equal to -1 the inequalities becomes

$$W(1)+W(2) > 1 \tag{2.14}$$

$$W(1) \qquad > 1 \tag{2.15}$$

$$W(2) > 1. \tag{2.16}$$

If W(1) and W(2) are chosen to be 1.5 the three inequalities will hold and the element will function as an "OR" gate.

$$1.5A + 1.5B > 1 \tag{2.17}$$

If negative weights are allowed an inverter can be implemented with a single weight equal to -1. Since the functions AND, OR, and INVERT can be implemented with perceptrons any logical function can be implemented with them.

It is very important to note that even though all logical functions can be implemented with perceptrons they cannot all be implemented in one level. It was demonstrated by Minsky and Papert [1] in their book "Perceptrons" that the EXCLUSIVE-OR function can only be implemented with two or more levels of perceptrons. Consider the following problem.

CASE 3

Design an "EXCLUSIVE-OR" function (C=A .XOR. B)

The equations for the perceptron are:

$$AW(1) + BW(2) + W(3) < 0 \tag{2.18}$$

$$AW(1) + W(3) > 0 \tag{2.19}$$

$$BW(2) + W(3) > 0 \tag{2.20}$$

$$W(3) < 0. \tag{2.21}$$

Combining (2.19) & (2.20) yields

$$AW(1) + BW(2) > -2W(3). \tag{2.22}$$

Combining (2.22) & (2.18) yields

$$AW(1) + BW(2) > -W(3) + AW(1) + BW(2) \tag{2.23}$$

$$0 > -W(3) \quad \text{OR } W(3) > 0. \tag{2.24}$$

Since it is known from Equation (2.21) that W(3) < 0, the problem will not have a solution with one perceptron.

The property of perceptrons not inability to perform the EXCLUSIVE-OR function in one level is of much importance historically and practically. This function is required in many problems. If the problems are complex it will be desirable to train the network rather than design it mathematically. Until recently it has been extremely difficult to train these multilevel networks. New advances in training using backpropagation have greatly enhanced the ability to train multilevel ANNs.

2.2 Types of Artificial Neural Networks

When studying the biological neural networks that comprise the cerebral cortex, it is found to be able to perform many functions. Among their basic functions are pattern recognition, sound recognition, speech comprehension, logic, associative memory, generalization, and interpretation of sensory input. [17] Different neuron models and different network configurations have been proposed to provide the best emulation of different neural functions.

Some of the most common network configurations are: Feedforward Networks, Hopfield Nets, Bidirectional Associative Memory, Adaptive Resonance Theory, and Counter Propagation. Some of these configurations require special training procedures or special neuron models. The weights in these networks are adjusted in one of two major ways:

(1) by calculation or

(2) by training.

Training is also divided into two main categories:

(1) supervised training and

(2) non-supervised training.

In supervised training known inputs are applied to the network and the outputs are observed. Adjustments are then made to the weights in order to change the prevailing outputs into the desired outputs. After many training sets have been applied and weights have been adjusted, the network will respond in the desired manner. In non-supervised learning various training sets are applied to the input of the network. As different features are recognized by the network it will adjust its weights so different input patterns can be categorized. When new input patterns are applied the nearest category to the input will be indicated by the output.

There is some concern among researchers as to whether certain network configurations or learning paradigms should be used based on how realistically they actually model biological neural networks. Some researchers believe that when designing a machine that emulates human brain functions, it is best to remain as close as possible to an accurate biological neural network. Others contend that as many liberties as are required should be taken in order to best solve the problem being studied. The latter philosophy will be used in the dissertation.

2.2.1 Feedforward Networks

Feedforward networks or nonrecurrent networks are one of the most common neural networks used today. A feedforward network consist of one or more rows of artificial neurons. The inputs to the first layer come from the sensory outputs. If more than one level is incorporated in the network design the outputs from each element of a previous level are fed forward into the inputs of each element of the next level.

The connections can be fully connected or sparsely connected. With fully connected networks each neuron takes its inputs from every neuron or sensor in the preceding level. In a feedforward network no neuron is allowed to feedback to a prior level. An example of a fully connected feedforward network is shown is Figure 7.

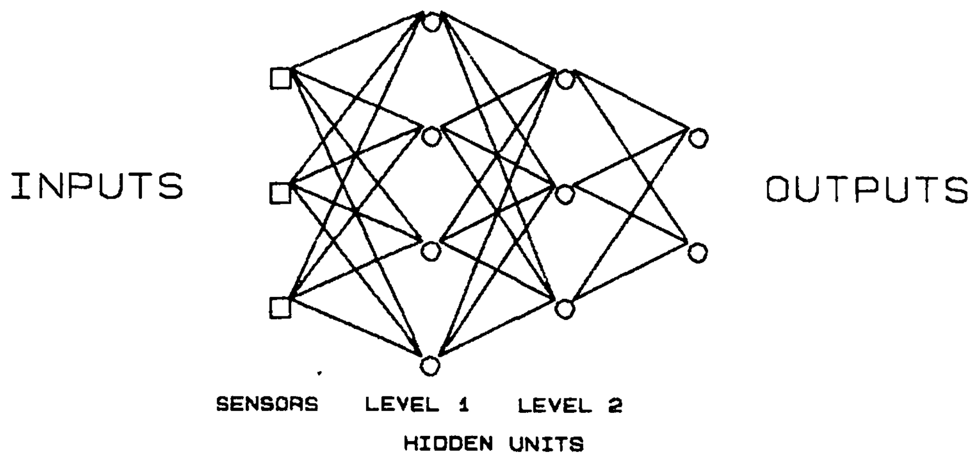


Figure 7

Fully Connected Neural Network

The neurons in the network that precede the output level are referred to as "hidden units". These hidden units can pose difficulties in training since their outputs cannot be measured directly. Feedforward networks have some important attributes. One of these attributes is their ability to be unconditionally stable. There is no feedback in the network. An input vector will simply propagate through the network and be mapped into an output vector. Since the network has no memory it cannot oscillate between states. The output of each neuron will be constrained by the nonlinear function $\frac{1}{1-e^{-x/T}}$ so the output will always be bounded by 1 and 0.

2.2.2 Recurrent Networks

Recurrent Networks, often called Hopfield Nets, are artificial neural networks which allow feedback loops. The main processing element is usually a perceptron. Unlike feedforward networks these networks are not guaranteed to be unconditionally stable. A typical example of a recurrent network is shown in Figure 8. In a recurrent network the output can maintain a "STATE". This state is simply the current binary vector that is represented by the network's output. Since its output state changes in response to changes in inputs and its current state, oscillations can occur. It has been shown by Cohen & Grossberg [15] that recurrent networks will be unconditionally stable if the weight matrix is symmetrical with all zeroes on the diagonal.

When this criterion is met the network is guaranteed to converge to a single stable state. The formula used to model a Hopfield network is [18]

$$C_i \frac{dU_i}{dt} = \sum_{j=1}^{P} T_{ij} V_j + I_i, \quad i = 1, 2 \ldots P$$

$$V_i = g(U_i). \tag{2.25}$$

Where C is the input electrical Capacitance of the operational amplifier used to model the artificial neuron.

V_i is the neuron output.

U_i= the neuron input.

T_{ij} are the weights (synapse strength).

(For the network to be guaranteed stable $T_{ii}=0$ and $T_{ij}=T_{ji}$).

g is a nonlinear function.

The energy of the network that converges to a minimum value is given by the Liapunov function [18]

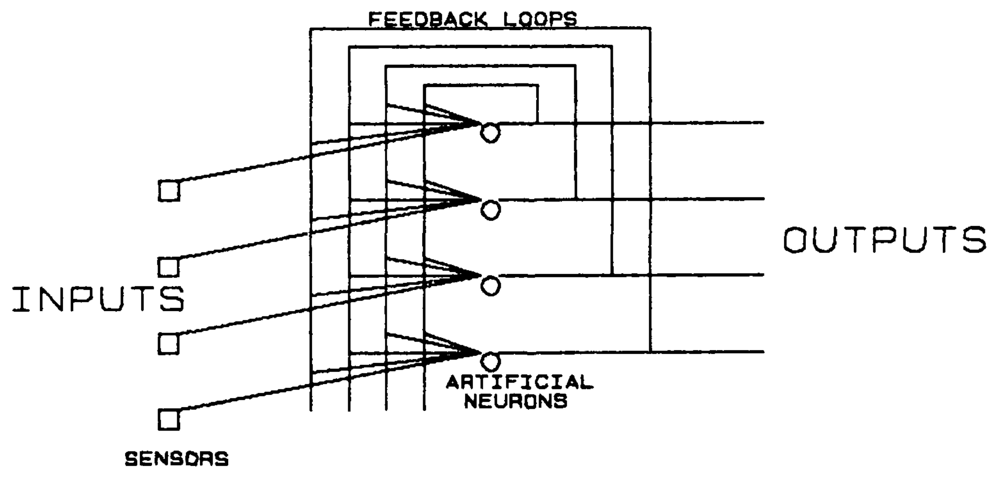


Figure 8

Recurrent Network

In Hopfield nets the weights are calculated and set rather than trained. There main usage is to find a good solution to optimization problems. They have been used as Analog to Digital Converters in which a network with an analog input common to all neurons will converge to a binary vector representing the signal's voltage. [19] Hopfield nets have also been used to provide a good solution to the Traveling Salesman Problem. [4] One of the advantages of Hopfield Nets is their ability to take a corrupted vector and find the best output vector associated with it. For this reason they are often referred to as Associative Memory.

2.2.3 Bi-directional Associative Memory

Bi-directional Associative Memory (BAM) like Hopfield Nets are recurrent networks. The BAM consists of two levels of neurons in which the second level is fed back into the input of the first level. The weight matrix of the first level must be the transpose of the weight matrix of the second level. An input vector can be inserted at the input of either level. The output of that level is then a new vector which is associated with the first. This output vector is then fed to the next level in the loop. Since the weight matrix of this level is the transpose of the previous weight matrix, the vector will be re-associated with the original vector which is the input in the first weight matrix. This will hold the network in a stable state. Since each vector is mapped back into itself the network will be unconditionally stable. Like the Hopfield Nets BAMs can take a partially corrupted vector and find the correct associations. But unlike the Hopfield Net it is bidirectional, in that it can accept either the vector or its associated vector and find the corresponding vector. An example of a BAM is shown is Figure 9.

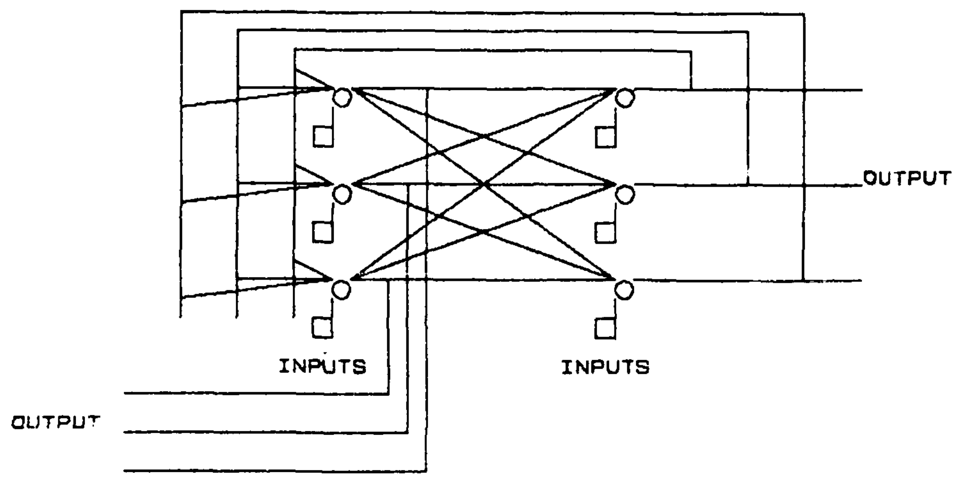


Figure 9

Model of BAM

2.2.4 Adaptive Resonance Theory

Adaptive Resonance Theory (ART) is a form of unsupervised learning. Instead of being trained with known vectors, random vectors are applied to the inputs of the network. They are compared with the memories of the network and classified accordingly. If no matches are found a new category is generated. This network model has the advantage of not requiring that an associated output vector be known for each input vector. It learns to classify vectors based on similarities recognized in the training set. This model is similar to biological neurons, in that it does not require a supervisor in order to learn. This ability to learn without supervision can be disadvantageous. In many cases the features being extracted from the input vector may not be the most obvious. In these cases supervised learning could provide a solution while ART would fail.

2.3 Training

Training is the method by which the weights of a neuron are modified in order to learn the correct network response. There are two major ways of training networks; statistical and deterministic. Deterministic training has evolved from the Delta Rule to more useful methods such as Backpropagation.

In order to train a neural network using supervised training, a set of known vectors must be input into the network and the outputs must be compared with the known results. Based on the correctness or error in the result the weights of the network are adjusted. Training can be a very time consuming process. One problem, that is common to both biological and artificial neural networks, is their tendency to forget. When a set of patterns are presented to a network and weights are adjusted to provide correct results, it is found that the first patterns will usually be forgotten by the time the later patterns are presented. This problem can be partially solved by repeatedly presenting all the patterns many times.

2.3.1 The Delta Rule

One of the most important training methods is the Delta Rule. With the Delta Rule the weights of each neuron are adjusted by using the following algorithm:

$$W(n+1) = W(n) + K\Delta X_{INPUT} \qquad (2.27)$$

$$\Delta = X_{TARGET} - X_{OUT} \, . \qquad (2.28)$$

where W(n+1) is the new adjusted weight.

W(n) is the old weight.

K is the training rate. (usually between 0.1 and 1.0)

X_{TARGET} is the known result that corresponds to the input.

X_{INPUT} is the input to the neuron.

X_{OUT} is the output response to input X_{INPUT} with weights W(n).

It can be noted from the above formula that when the error term (X_{TARGET} - X_{OUT}) is zero, no weight modification will occur. When this term is not zero, training will occur in the direction and strength that is proportional to it. It should also be noted that when no input from X_{INPUT} excites the neuron no modifications are made to the weight associated with this input. This means that if this input did not contribute to the error, its weight should not be modified. This algorithm can be used to train perceptrons in single level networks or multilevel networks in which desired outputs for each level are known. However, multilevel networks in which the outputs for the hidden units are not known present serious problems. In 1969 Minsky & Papert proved in their book "Perceptrons" [1] that many important functions such as EXCLUSIVE-OR or Parity could not be solved in a single level of perceptrons. This combined with the difficulties encountered in training multilevel networks did much to deter research on neural networks during the 1970's. However in 1986 several researchers, Rumelhart, Hinton, and Williams developed a training method for training multilevel networks known as backpropagation or the Generalized Delta Rule. Backpropagation was also discovered by Werbos in 1974, and Parker in 1982, but a book by Rumelhart and McClelland [3] gave a particularly detailed explanation of these training rules. This discovery has done much to rekindle interest in neural networks in the 1980's.

2.3.2 Backpropagation

Backpropagation or the Generalized Delta Rule provides a method of training all of the weights in a multilevel feedforward neural network. The method of training the weights is a gradient descent method. The change in the error of the network is calculated with respect to each weight. This value is the slope of an error curve like the example shown in Figure 10.

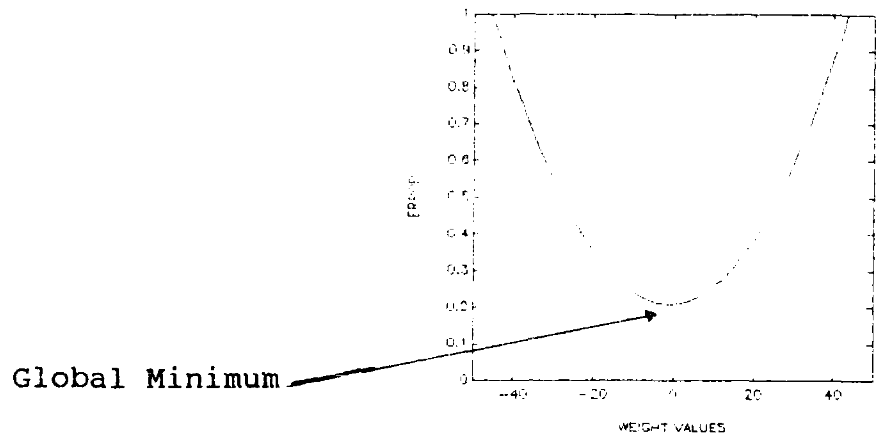


Figure 10

Error Curve

The weights are then adjusted by an amount that is proportional to the negative slope of the curve. The formula for the error in the network for the final level in response to N patterns is

$$E_{TOTAL} = \sum_{P=1}^{N} E_P \qquad (2.29)$$

$$E_P = (T_P - O_P)^2. \qquad (2.30)$$

where E_{TOTAL} is the total error from the N patterns,

E_P is the error from pattern P,

T is the target output value,

O is the output value.

It should be noted that when only one pattern is presented the total error will equal $(T-O)^2$, which will form a parabolic curve similar to the example in Figure 10. This parabola has a minimum value known as the "global minimum".

The equations modeling neuron (j) are

$$O_j = f(NET_j) \tag{2.31}$$

$$NET_j = \sum_{i=1}^{N} w_{ij} O_i . \tag{2.32}$$

where w_{ij} is the weight connecting the output of neuron (i) to neuron (j).

NET is the internal summation of the weighted inputs to neuron (j).

and f is the nonlinear function $\frac{1}{1+e^{-x}}$.

For the output level of the network the error slope can be calculated as follows.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial \sum E_P}{\partial w_{ij}} = \sum \frac{\partial E_P}{\partial w_{ij}} \tag{2.33}$$

By using the chain rule the derivative of the error due to one pattern with respect to any weight can be expressed as:

$$\frac{\partial E_p}{\partial W_{ij}} = \frac{\partial E_p}{\partial O_j} \cdot \frac{\partial O_j}{\partial NET_j} \cdot \frac{\partial NET_j}{\partial W_{ij}} \tag{2.34}$$

Carrying out these differentiations will yield:

$$\frac{\partial E_p}{\partial W_{ij}} = -2(T - O) \cdot f'(NET) \cdot O_i. \tag{2.35}$$

The derivative of the nonlinear function $f(x) = \frac{1}{1 - e^{-x}}$ is

$$f'(x) = f(x) \cdot (1 - f(x)). \tag{2.36}$$

So the formula for adjusting the weights is

$$\frac{\partial E}{\partial W_{ij}} = -2 \cdot (T_j - O_j) \cdot (f(NET)) \cdot (1 - f(NET)) \cdot O_i. \tag{2.37}$$

Since the correction to weight W_{ij} is proportional to the negative derivative of the error, the correction formula will be

$$\Delta W_{ij} = K \cdot (T_j - O_j) \cdot f(NET) \cdot (1 - f(NET)) \cdot O_i. \tag{2.38}$$

Where K is the learning rate.

The real power of backpropagation is its ability to train the hidden units of the network. The training equation must be modified since the target values, T_j, are not known for hidden units. First δ_{pj} is defined as $\frac{\partial E_p}{\partial NET_{pi}}$. In terms of the output units it can be expressed as

$$\delta_{pi} = -2 \cdot (T - O) f'(NET). \tag{2.39}$$

The training formula is $\Delta W_{ij} = K \cdot \delta \cdot O_i$. From the chain rule the definition of δ_{pi} can be modified to be

$$\delta_{pi} = \frac{\partial E_p}{\partial O_{pj}} \cdot f'(NET_{pj}). \tag{2.40}$$

By using the chain rule again, $\frac{\partial E_p}{\partial O_{pj}}$ can be rewritten in terms of the previous level's NETs

$$\delta_{pi} = \sum_K \frac{\partial E_p}{\partial NET_{pk}} \cdot \frac{\partial NET_{pk}}{\partial O_{pi}} \cdot f'(NET_{pj}) \tag{2.41}$$

OR

$$\delta_{pi} = \sum_K \frac{\partial E_p}{\partial NET_{pk}} \cdot W_{kj} \cdot f'(NET_{pj}). \tag{2.42}$$

From the definition of δ for an arbitrary level the formula can be rewritten as

$$\delta_{pj} = \sum^{k} \delta_{pk} \cdot W_{kj} \cdot f'(NET_{pj}) \tag{2.43}$$

OR

$$\delta_{pj} = f'(NET_{pj}) \sum^{k} \delta_{pk} \cdot W_{kj}. \tag{2.44}$$

The training formula for hidden units is then

$$\Delta W_{ij} = K \cdot \delta O_i = K \cdot O_i \cdot f'(NET_{pk}) \sum_K \delta_{pk} \cdot W_{kj}. \tag{2.45}$$

The weights are adjusted by adding the correction to the current weight values.

$$W_{ij}(n+1) = W_{ij}(n) + \Delta W_{ij} \tag{2.46}$$

The weight changes are often modified by adding a momentum term. The momentum term acts as a low pass filter to limit oscillations in the training session. It is applied to the equation as follows.

$$\Delta W_{ij} = K \cdot \delta \cdot O_i + \Delta W_{ij} \cdot M \tag{2.47}$$

Where M is the momentum term.

The choice of a momentum term, M, and a learning rate term, K, is very important when training a network. If K is too small the training can take a tremendous amount of time to converge. If K is too large oscillations can occur and prevent conversion. Extreme values of M can also affect the time of convergence. Good values for these constants are usually 0.1 to 1.0. The best values to which the weights should be initialized are small random values. Care should be taken to ensure that all weights are not set to the same value. If they are set to the same value and a fully connected network is used, all weight change calculations will yield the same value and all weights will remain equal to one another throughout the training session. During a training session the weights could possibly fall into a state with all equal wieghts, but this would be extremely rare. Adding extremely small amounts of noise to the weights after each training iteration can prevent this state from developing.

There is a very important problem that often counteracts the improvements gained by using backpropagation. This problem is the ability to fall into a "local minimum". These "local minima" are locations on the error curve where the slope in all dimensions is 0. The network might find one of these and represent this as the correct solution. An example of one of these error curves with several local minima is shown in Figure 11.

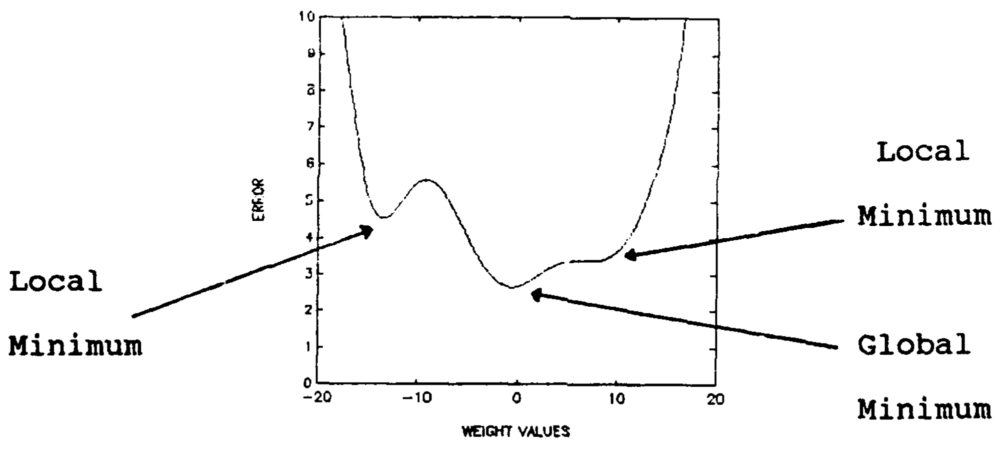


Figure 11

Error Curve with Local Minima

As was shown in Equations (2.29) and (2.30) the error curve will be a parabola when one test pattern is used, but when multiple test patterns are used an error curve composed of the summation of multiple parabolas will result. It is the summation of these curves that form the local and global minima like the ones shown in Figure 11.

A more detailed discussion and derivation of the backpropagation algorithm can be found in "Parallel Distributed Processing" [3] by Rumelhart and McClelland.

2.3.3 Statistical Training

Statistical training is a method of randomly adjusting weights that eliminates many of the local minima. The basic procedure is: [4]

(1) Place a vector on the input of the network.

(2) Measure the total error in the output.

(3) Select a weight at random.

Change its value a small random amount.

(4) Re-measure the error.

(5) If the error is decreased keep the change.

Otherwise retain the original weight.

(6) Repeat the procedure until a suitable solution is reached.

This procedure is analogous to the annealing of metals in the way it moves from a high energy state to a minimum energy state. Therefore it is often referred to as "simulated annealing". Another form of statistical learning is known as Boltzmann's Training. In this form the changes, C, are determined by the formula

$$P_C = e^{-C/KT} . \qquad (2.48)$$

where K is the Boltzmann constant.

T is an artificial temperature determined by

$$T(t) = T_0/\log(1+t) . \qquad (2.49)$$

T_o is the initial temperature,

and t is time.

2.4 Linearly Separable Regions

It has been demonstrated in an earlier section how the perceptron can be used to perform simple logic by using one level of perceptrons. These perceptron equations with variables A and B take the form:

$$W_1A + W_2B + W_3 > 0 \quad \text{or} \tag{2.50}$$

$$W_1A + W_2B + W_3 < 0. \tag{2.51}$$

The equation $W_1A + W_2B + W_3 = 0$ describes a straight line. The perceptron therefore can be used to separate points above the line from those below it. If three inputs are used in the perceptron this line will become a plane in three dimensional space. If N inputs are used it will be expanded into N dimensions and define an N-1 dimensional hyperplane that divides an N dimensional hypervolume. When additional levels of perceptrons are used the complexity of the pattern being distinguished can be increased. A two-level network for example can be used to separate any convex region from its background. Three levels of neurons can be used to separate any regions regardless of their geometry. Consider the following example shown in Figure 12.

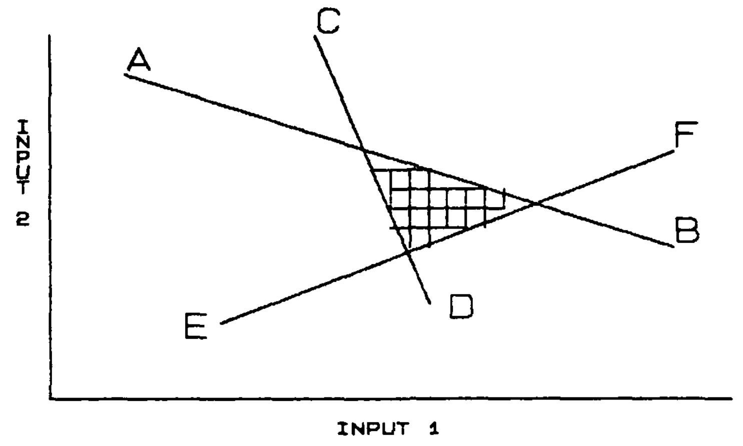


Figure 12

Example of Linearly Separable Region

One perceptron could be used to separate points above line AB, another to separate points to the left of line CD and another to separate points below EF. If a second level of perceptrons is used to "AND" the outputs of the first three perceptrons, the convex region can be separated from all other points. An example of a convex region is shown in the shaded area of Figure 12. Any region or set of regions can be divided into a set of convex regions. A third level of perceptrons can then be used to "OR" together the convex regions and thus separate any regions from all other points.

This provides a very powerful tool for use in pattern recognition problems. With this ability a network can be designed and trained to separate any pattern from its background or another pattern. The network will be unconditionally stable and its output will be bounded by 1 and 0. It can be implemented in only three levels of perceptrons and will operate in realtime with a propagation delay of only three delay units. Where a delay unit is the propagation delay per level of perceptrons.

There are however several major problems with this form of network:

(1) Training times can be extremely long when backpropagation is used.

(2) The number of hidden units required to work difficult geometries can be very high.

(3) The training solution can fall into a "local minimum".

Chapter 3

Beamformer Fundamentals

3.1 Basic Types of Beamformers

Beamformer arrays are used in a variety of different applications. The array can be of any size, but the basic operation is usually one of two functions. Either the information received by the array is processed in order to indicate the direction from which a wave is approaching or if the direction is already known, the information is processed by steering the beamformer in the direction oi the wave. If the wave is steered toward the source of the signal, noise from other sources will partially cancel and an improved signal to noise ratio of the signal of interest will be realized.

3.2 Narrowband Beamformer

When a wavefront approaches an array of sensors like the one shown in Figure 13, the array will sample the wave in space as well as in time. If instantaneous measurements are recorded on all N sensors, and if the wave is a single frequency sinusoid the resulting vector can be represented by the following equation

$$V_i = \mathrm{SIN}(\omega t + \alpha i). \qquad (3.1)$$

$$\text{where } \alpha = \frac{2\pi \cdot d \cdot \mathrm{SIN}(\theta)}{\lambda}$$

and $2\pi i$ is the wave number.

ω is the waves frequency.

d is the distance between sensors.

λ is the wavelength.

θ is the arrival angle

and t is time.

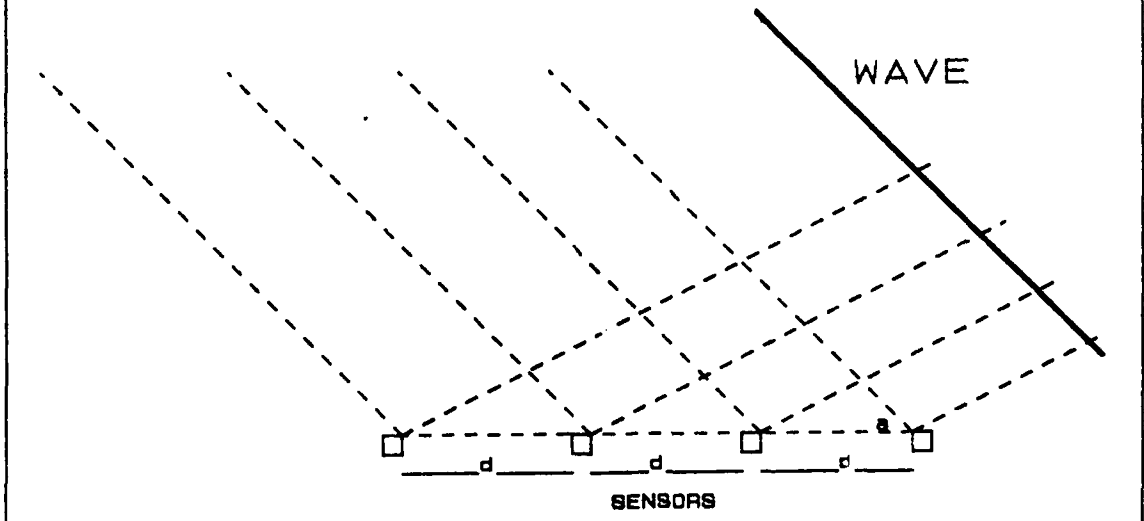


Figure 13

Beamformer Array

Therefore at any instantaneous point is time with the signal frequency held constant, the output, V_i, viewed over space will be a sinusoid whose phase is determined by the arrival angle, θ, the distance between sensors, d, and the wavelength, λ.

The angle of approach, θ, can be determined by analyzing the space sampled information. The output of this analysis will be a function of both the angle of approach, the frequency of the wave being studied, and the propagation speed of the media through which the wave is traveling. If the propagation speed and the wave frequency are known, the angle of approach can be determined from the space sampled data.

One implementation often used to process the output of a beamformer array consist of an operational amplifier used to sum the output. If N sensors are summed the resulting output will be:

$$OUTPUT = \sum_{i=0}^{N-1} U_i = \sum_{i=0}^{N-1} SIN(\omega t + \alpha i) \tag{3.2}$$

$$OUTPUT = \frac{SIN\left(\frac{N}{2}\alpha\right)}{SIN\left(\frac{1}{2}\alpha\right)} \cdot COS\left(\omega t - \frac{(N-1)}{2}\alpha\right) \text{ for } \alpha \neq 0$$

$$OUTPUT = N \cdot SIN(\omega t) \text{ for } \alpha = 0. \tag{3.3}$$

The amplitude of this output is the term

$$A = \frac{SIN\left(\frac{N}{2}\alpha\right)}{SIN\left(\frac{1}{2}\alpha\right)} \text{ for } \alpha \neq 0$$

$$A = N \text{ for } \alpha = 0 \tag{3.4}$$

This output will be a maximum when $\alpha = 0$ (when the wave is parallel to the line of sensors and perpendicular to the boresight). This is known as the steered direction. Depending on the number of sensors, N, the output may have several sidelobes which can allow noise from directions other than 0 degrees to interfere with the received signal.

If it is desired to know the direction from which a signal propagates, multiple beamformer arrays could be used. These arrays could each be aligned in the direction perpendicular to the direction it was designed to observe. The array with the highest output would indicate the direction from which the wave was approaching. A more practical method would be to process the information from the same array through multiple operational amplifiers each of which is designed to respond to a different angle of arrival. This steering can be accomplished by placing an appropriate time delay at each input to the operational amplifier so the summation equation will be

$$V = \sum_{i=0}^{N-1} \mathrm{SIN}(\omega(t-\Delta_i)+\alpha i). \tag{3.5}$$

Where Δ_i is the time delay.

If $\omega\Delta_i$ is equal to αi the beamformer will be steered toward angle θ. Since $\alpha = \frac{2\pi d\,\mathrm{SIN}(\theta)}{\lambda}$ the delay should therefore be set for

$$\Delta_i = \frac{2\pi i d \mathrm{SIN}(\theta)}{\omega\lambda} \tag{3.6}$$

$$\Delta_i = \frac{i d \mathrm{SIN}(\theta)}{\mathrm{PROPAGATION\ VELOCITY}}. \tag{3.7}$$

Another way of accomplishing beamformer steering is by multiplying the signal by $e^{-j\alpha i}$ in order to shift it by the appropriate phase. (i is the sensor index and j is $\sqrt{-1}$).

$$Y = \sum_{i=0}^{N-1} [\mathrm{SIN}(\omega t + \alpha i)] e^{-j\alpha i} \tag{3.8}$$

$$Y = \sum_{i=0}^{N-1} \frac{e^{-j2\alpha i}}{2} [\mathrm{SIN}(\omega t) + j\mathrm{COS}(\omega t)] \tag{3.9}$$

The phase of the sine wave will no longer be a function of the wavenumber, α. Therefore the steering can be accomplished by multiplying the sensor's output by the weight $W_i = e^{-j\alpha i}$. This phase adjustment seems to be the simpler of the two methods but the weight's value will be a function of ω. This will not present a problem if ω is a constant, but if ω varies and covers a large bandwidth additional processing will be required.

3.3 Wideband Beamformers

When the operation of a beamformer over a large bandwidth is desired, changes to the beamformer processing must be made. Since the frequency of the wave is not constant the resulting sensor output will be a function of ω as well as θ.

$$V_i = \mathrm{SIN}(g(\omega,t) + f(0,i)) \qquad (3.10)$$

From equation 3.1

Implementation of wideband beamformers is sometimes accomplished by bandpass filtering the output of each sensor into narrow bands and then processing each band through a narrowband beamformer. An example is shown in Figure 14.

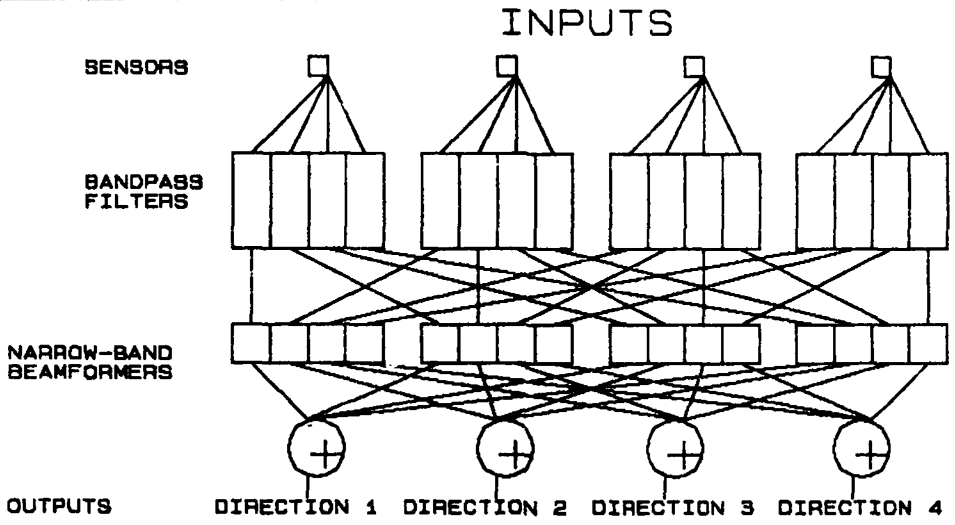


Figure 14

Wideband Beamformer

When the direction of a wavefront is known and it is desired to focus on the signal source, steering can be accomplished by placing time delays in the lines. This can be performed just as was done with narrowband beamformers. However, a phase delay cannot be used to shift the phase since the phase is a function of frequency and the frequency is not constant. In order to process these wideband signals they must be filtered into narrow bands and processed separately. One novel method of implementing this filtering and subsequent phase shifting has been reported by Follett [5]. This method incorporates the use of Discrete Fourier Transforms or Fast Fourier Transforms to process the digitized sensor outputs.

3.4 Previous Work

An extensive amount of literature has been published over the past two decades on beamformer arrays. Specialized systems for seismic, sonar, radar, and radio telescope arrays have been reported and analyzed. [16] Three of these systems which represent the progress in beamformer arrays and Neural Networks are described below. First one system using an FFT beamformer is described. Another form of these beamformers which uses the earliest form of artificial neuron, the ADALINE, is also discussed. Lastly some more recent research combining both beamformers and Hopfield Networks is presented.

3.4.1 Fast Fourier Transform Wideband Beamformer

One method of processing wideband beamformer arrays is with a Fast Fourier Transform (FFT) Beamformer. [5] With such a system an FFT or Discrete Fourier Transform (DFT) is performed on the output of the sensor data. This first DFT filters the data into narrowbands. The output of each frequency band is then routed to another Fourier Transform where the space sampled data is transformed into a domain which represent the wave number of the space sampled signal. It should be noted that the equation used to add the phase delay to the sensor outputs is

$$Y_{\alpha} = \sum_{i=1}^{N-1} X_i e^{-j\alpha i}. \qquad (3.11)$$

This equation is mathematically equivalent to the Discrete Fourier Transform which is

$$F_k = \sum_{i=0}^{N-1} f_i e^{-jw_k i}. \qquad (3.12)$$

If the number of time samples and the number of sensors are a power of two an FFT can be used to calculate these shifts. The data can then be re-routed to an Inverse Fourier Transform where it can be transformed back into the time domain.

To demonstrate the use of Fourier Transforms in processing beamformer arrays, the angle response of a narrow band signal was plotted for 9 bands. The DFT was evaluated at equal phase increments which translated into arrival angles of 0.0, 7.18, 14.48, 22.02, 30.00, 38.68, 48.59, 61.04, and 90.00 degrees. These values were calculated from Equation (3.1).

$$\alpha = \frac{2\pi d \, \mathrm{SIN}(\theta)}{\lambda} \qquad (3.13)$$

or

$$\theta = \mathrm{SIN}^{-1}\left(\frac{\alpha\lambda}{2\pi d}\right) \qquad (3.14)$$

If the sensor distance, d, is set equal one half the wavelength, $\frac{1}{2}\lambda$, the formula reduces to $\theta = \mathrm{SIN}^{-1}\left(\frac{\alpha}{\pi}\right)$. As the phase varies from 0 to π the arrival angle, θ, will vary from 0 to 90 degrees.

The bandwidth can be determined by the following formula taken from Follet [5].

$$BW_{3\text{-}DB} = .886\left(\frac{\lambda}{L}\right)\mathrm{Sec}(\theta),\quad L \text{ is array length} \qquad (3.15)$$

or

$$BW_{3\text{-}DB} = 2\sqrt{.886\left(\frac{\lambda}{L}\right)},\quad \text{near endfire} \qquad (3.17)$$

A table that shows the relationship between phases arrival angles and bandwidth is shown is Table 1.

Phase Degrees	Arrival Angle Degrees	Bandwidth Degrees
0	0.00	5.08
20	6.38	5.11
40	12.84	5.21
60	19.47	5.38
80	26.39	5.67
100	33.75	6.11
120	41.81	6.81
140	51.06	8.08
160	62.73	11.08
180	90.00	34.00

Table 1

Bandwidth of Arrival Angles

It should be noted that the spacing of d was set equal to one half the wavelength of the wave. This wavelength is a function of the frequency of the wave and the velocity at which it travels. This selection of sensor spacing, d, tunes the array to work at a specific frequency. If wideband signals are to be detected, modifications to the system must be made. One solution to this problem can be achieved by using multiple arrays of sensors known as sub-arrays. The output of each sensor in each array is converted into the frequency domain with a DFT or FFT. When the data is routed to the next level of the FFT beamformer only the frequencies corresponding to the sub-array's tuned frequency will be included in the routing.

There are several problems that should be noted with the DFT method of processing. First there are sidebands that could be mistaken for the main signal. Secondly, when equal phase increments are used, (which is a requirement for FFTs) the resulting arrival angles will be defined sharply for lower angles and very coursely for higher angles. When the wave direction is not known it will be more useful to have equal arrival angle detection. An FFT narrowband beamformer example is shown in Figure 15. The angle response of a wideband beamformer is shown in Figure 16. In this example 20 sub-arrays were used to analyze data with a bandwidth of 30 to 300 Hz. This wideband example is clearly inferior to the narrowband example. FFT beamformers have several important advantages. They have the capability of detecting signals from different

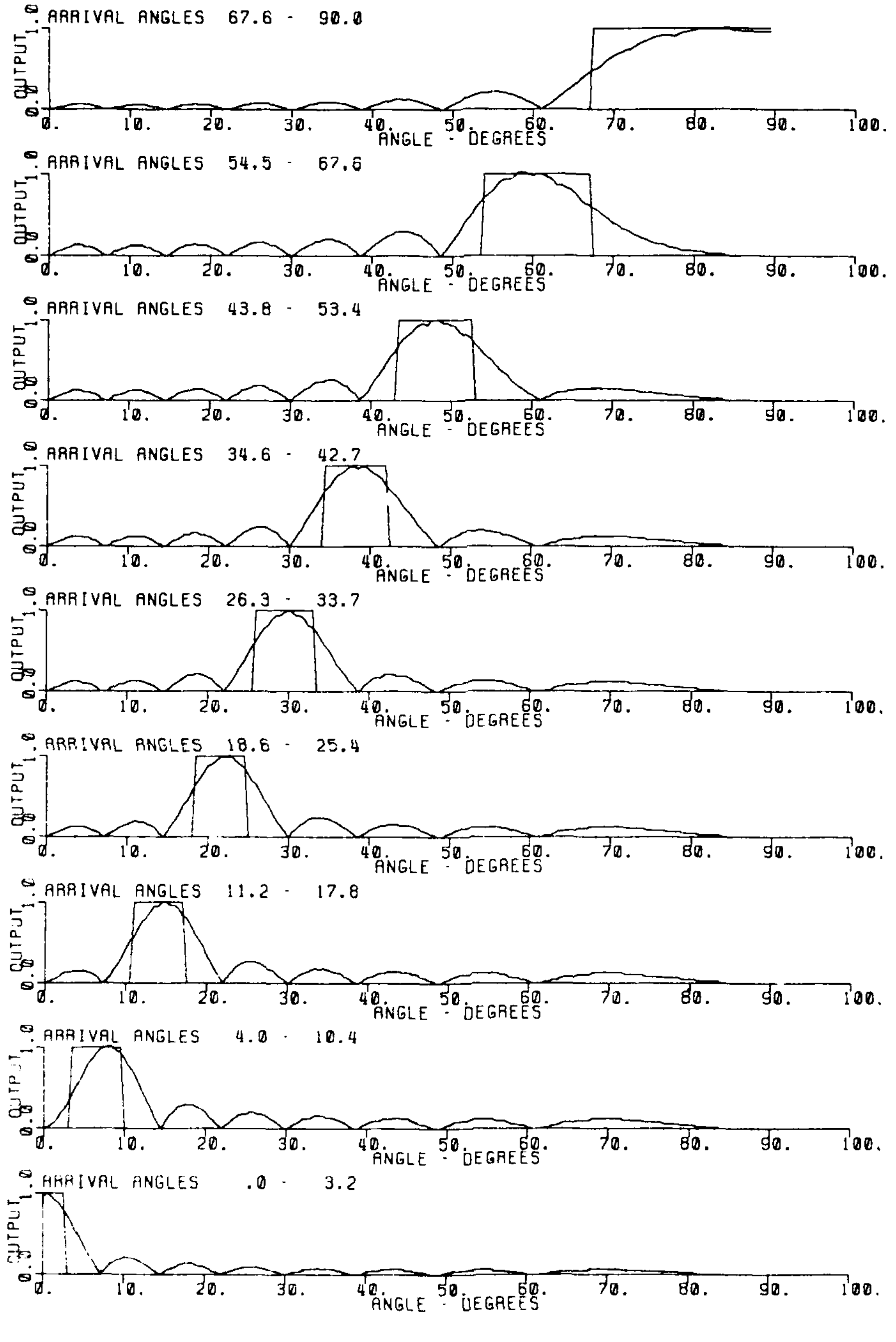


Figure 15

DFT Narrowband Beamformer Response

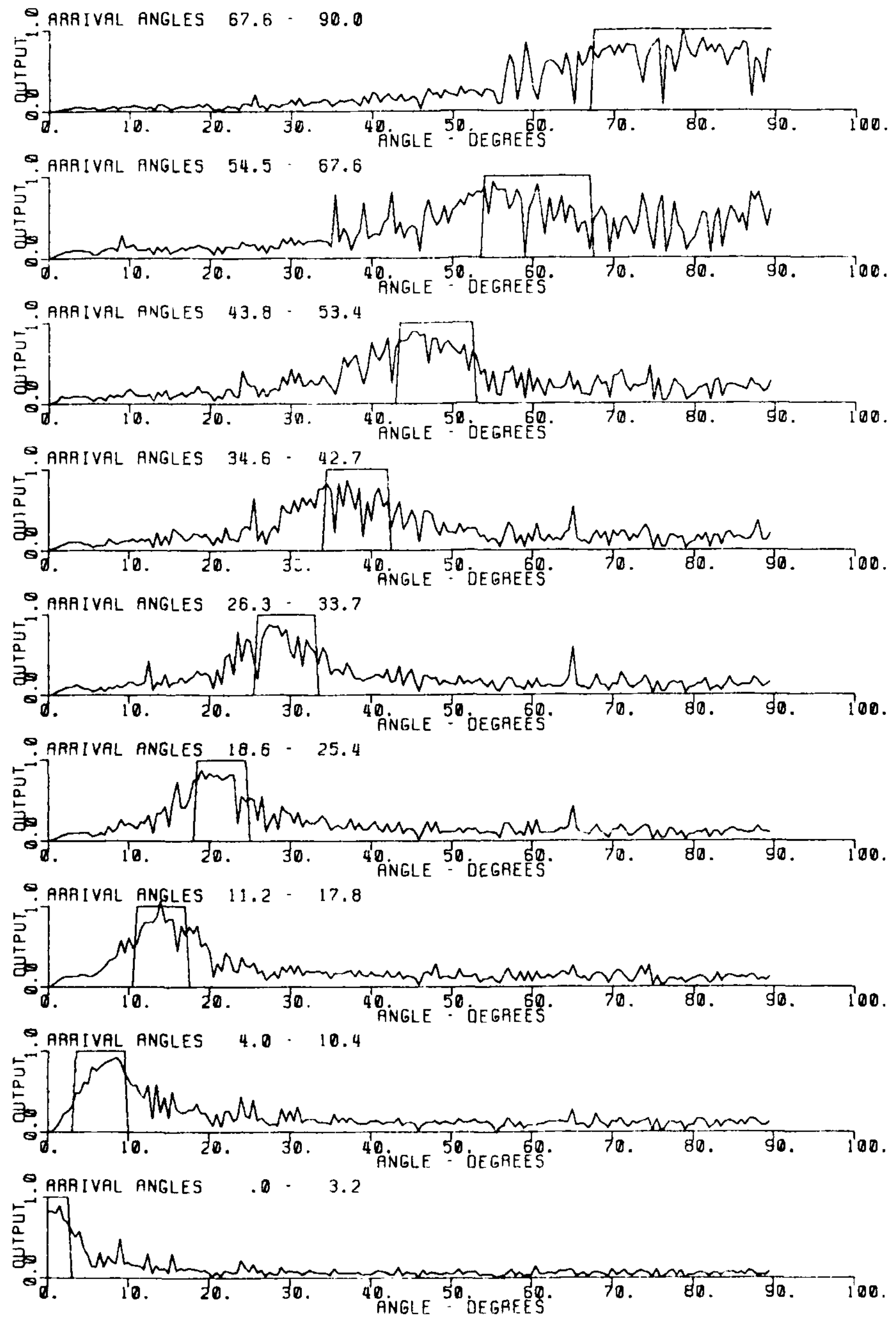


Figure 16

DFT Wideband Beamformer Response

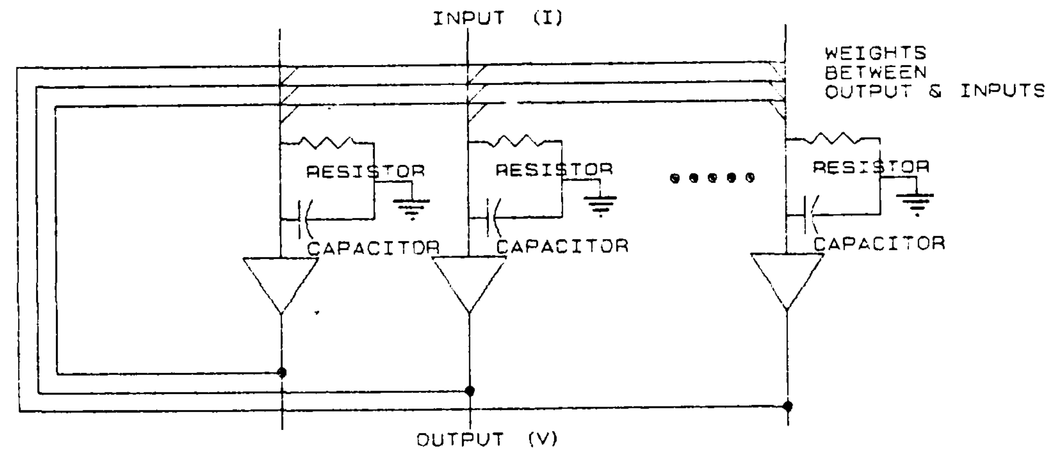
directions concurrently. They also can provide a time history output. This can be of great importance in determining not only the direction of the signal but its signature as well. The FFT beamformer plots are presented with the appropriate 3 db passband superimposed on each plot. The 3 db passband is the band between the half power points. These half power points are the locations where the signal is attenuated by a factor of 0.707.

3.4.2 Adaptive Beamformers with ADALINES

Artificial Neural Networks have been used in the form of ADALINES in numerous applications. Many of these have been reported by Widrow, the inventor of the ADALINE, and Stearns. [7] The ADALINE is primarily an adaptive filter. In beamformer arrays they can be used to adapt for noise cancelation or array steering. In many of these applications the least-mean-square (LMS) algorithm is used. This is the algorithm on which the Delta Rule was based. In this algorithm the square of the error is minimized by adjusting weights on the ADALINE until an optimum filter has been adapted. A pilot signal is often used to provide a source with which to train the network. These networks provide an analog output just as the FFT beamformer method does. When a pilot signal is used to train the network, it will allow the network to adapt to interference or noise signals that can be unique to a particular environment.

3.4.3 Beamforming with Hopfield Networks

Methods of using Hopfield Networks to determine the angle of arrival have been reported by Park [20], Rastogi, et al [21], and Goryn and Kaveh [28]. A summary taken from their work follows. In these efforts the wave number of a narrowband signal or the frequency of a time sampled signal can be determined. Multiple frequencies can be determined with the network shown in Figure 17.



HOPFIELD AND TANK NEURAL NETWORK MODEL
(FROM GORYN & KAVEH [28])

Figure 17

Hopfield Network for Beamforming

In this network the input is a function of frequency, phase, and amplitude. The output is a 1 or a 0, which corresponds to the presence or absence of a signal in the form: (taken from Rastogi's derivation [21])

$$S_i = a_i e^{j\theta_i}[e^{j\omega_i}, e^{j2\omega_i}, \dots, e^{jN\omega_i}]^T \text{ for frequency detection} \quad (3.17)$$

or

$$S_i = a_i\left[1, e^{j2\pi\frac{d}{c}\mathrm{SIN}(\phi)f_i}, \dots, e^{j(M-1)2\pi\frac{d}{c}\mathrm{SIN}(\phi)f_i}\right] \quad (3.18)$$

for arrival angle detection.

The error to be minimized in the network is

$$E = \| y - [S_1, S_2, \dots S_P]V \|_2^2. \quad (3.19)$$

Where y is the analog input

S_i is the signal vector described above.

V is the binary output vector.

Manipulation of this formula can be shown to yield

$$E = y^t y + V^T S^t S V - y^t S V - V^T S^t y. \quad (3.20)$$

Since Equation (3.20) is being minimized with respect to V, terms without V can be removed.

$$E = V^T S^t S V - 2 \cdot \Re(y^t S V) \quad (3.21)$$

Since $T_{ii}=0$ for stability the term must be subtracted

$$E = V^T S^t S V - 2 \cdot \Re(y^t S V) - \sum_{i=1}^{P}((S_i^t S_i)V_i(V_i - 1)). \quad (3.22)$$

This equation can be further manipulated to become

$$E = \Re\left[\sum_{i=1}^{P} \sum_{\substack{j=1 \\ j<>i}}^{P} 2(S_i^t S_j)V_i V_j - \sum_{i=1}^{P}(2y^t S_i + S_i^t S_i)V_i \right]. \quad (3.23)$$

(t is the transpose conjugate & T is the transpose) Comparison of this formula with the energy function for the Hopfie'd Net,

$$E = -\frac{1}{2}\sum_{i=1}^{P}\sum_{j=1}^{P} T_{ij} V_i V_j - \sum_{i=1}^{P} I_i V_i. \qquad (3.24)$$

shows that the feedback weights, T, should be equal to

$$T_{ij} = -2\Re[S_i^t S_j] \qquad (3.25)$$

and the input weight should be

$$I_i = \Re\left[y^t S_i - \frac{1}{2} S_i^t S_i \right]. \qquad (3.26)$$

The weights for the feedback can be calculated from the known values of S_i. However the weights for the inputs must be calculated from the input vector and the S_i vector. When the network converges, each output neuron will take on an output value of 1 or 0 indicating the presence or absence of a wave of the corresponding amplitude, phase, and frequency. Simulations made by Rastini, et al [21] were very successful in detecting the input's frequency spectrum when the SNR was 5 db.

This network has a very important advantage when used to detect angles of arrival, in that it can detect multiple arrival angles. The network can also provide very fine accuracy in the angle being resolved. There are however several important disadvantages. One is the problem of setting the weights for the network. The inputs weights are a function of the input, y_i. This means that the vector multiplication between y and S must be performed for each input set. The network also requires a large quantity of neurons. One for each combination of angle, phase, and amplitude to be detected. Park and Rastogi have both reported ways of reducing this number of neurons, but considerable preprocessing is still required. Goryn and Kaveh have extended the procedure to include wideband signals, but preprocessing is still required including a DFT on the input data. An example of the angle response of a narrowband beamformer designed with an Hopfield net is shown in Figure 18. This network was designed using 50 gages and 90 neurons for 90 evenly spaced arrival angles. The network was tested by generating a signal of each angle and processing it through the network with 12 iterations of the following formula.

$$u_i(k+1) = u_i(k) + \sum_{j=1}^{P} T_{ij} v_j + I_i \qquad (3.27)$$

$v_i = f(u_i)$; f() is the nonlinear function.

This formula is a discrete version of Equation (2.25) which models the Hopfield network.

In Figure 19 an example is shown in which three different arrival angles are presented to the network. These angles were 10, 50, and 62 degrees. After 6 iterations all three angles were correctly identified.

When training examples can be submitted to a network in order to teach it the proper response, an important improvement can be realized. All of the beamformer algorithms presented so far consider a situation in which the propagation speed is constant and the wave is approaching from one direction. This may be accurate for some applications but for many others the propagation speed will vary if the media through which it travels is not homogeneous. Reflections and echos are of a considerable consequence in both seismic and sonar applications. To address these problems more thoroughly feedforward networks can be trained with known signals to provide beamformers which adapt to the environmental characteristics and provide a more accurate response.

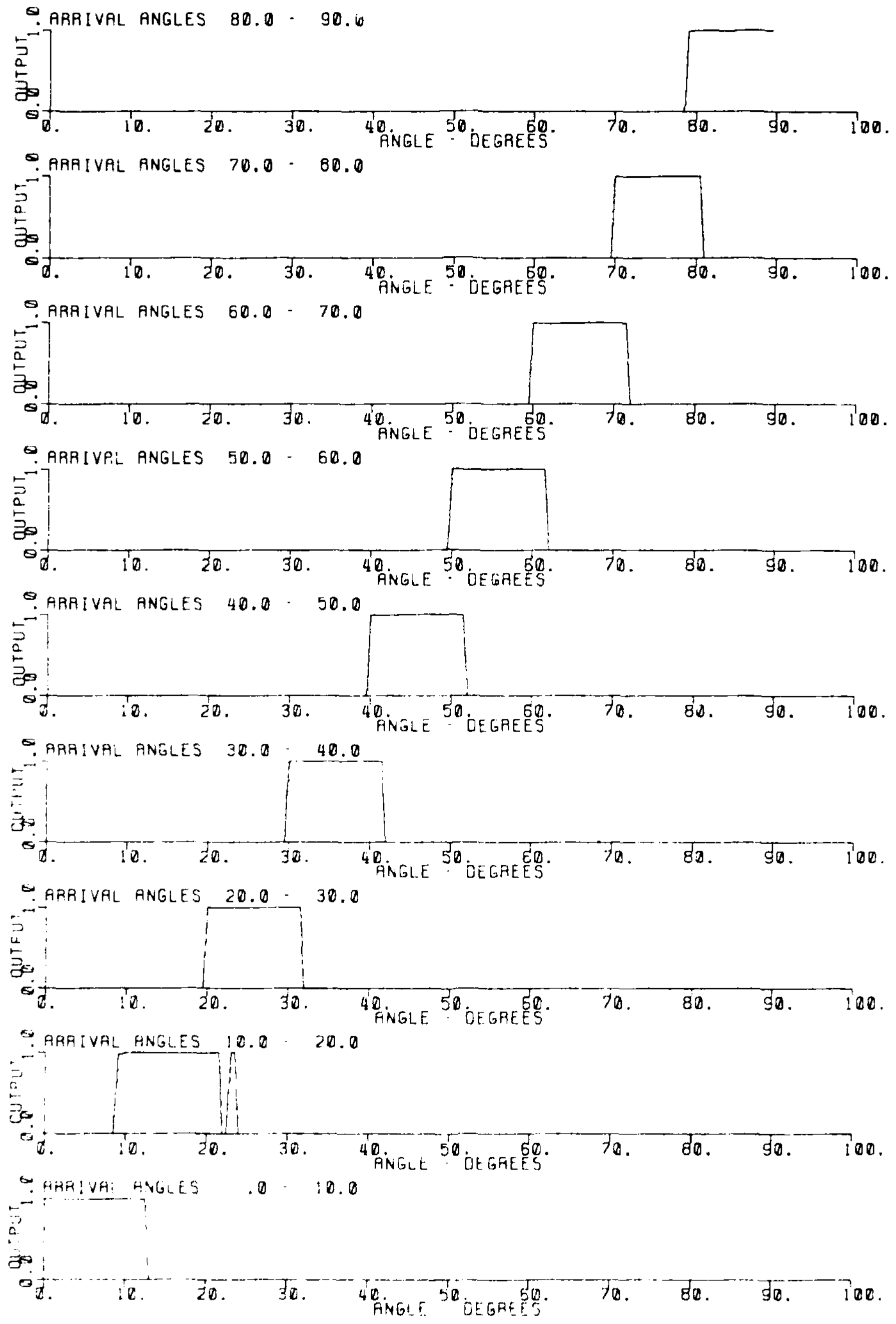


Figure 18

Angle Response of Hopfield Net ANN

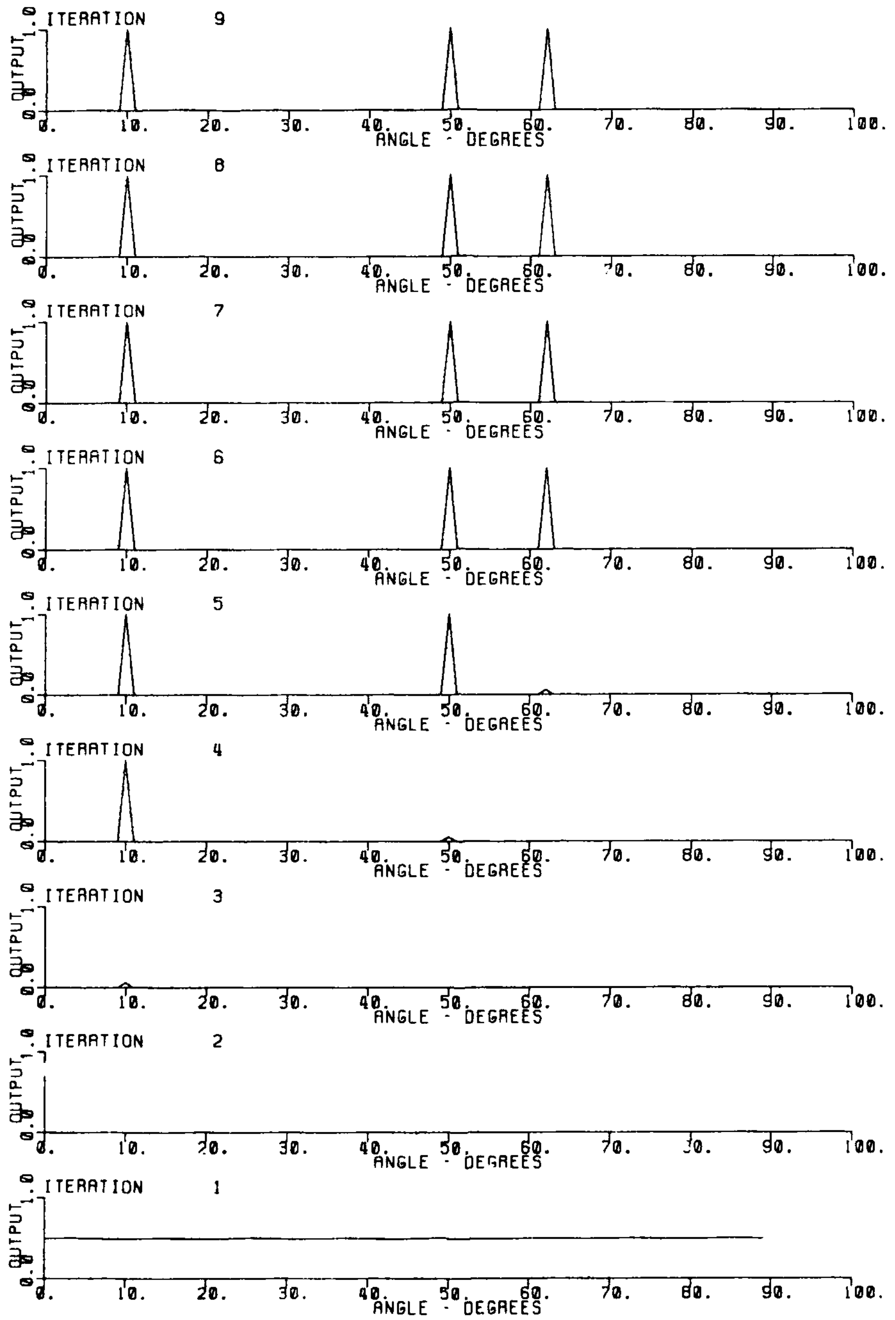


Figure 19

Hopfield Network Detection of 3 Angles

Chapter 4

Model of Narrowband Beamformer

4.1 Beamformer Array Processing with ANNs

Artificial Neural Networks provide an interesting way of processing the information received from beamformer arrays. When the desired application is one of determining the direction of the received signal, ANNs offer several advantages over digital signal processing networks. Rather than requiring special processing modules such as analog to digital converters or special digital signal processing circuits the ANN provides a system which will accept the sensor's analog output and produce a digital output that represents the presence or absence of a wave received from a designated direction. ANNs operate in realtime with very small propagation delays. However, ANNs lack the precision and qualitative results that can be provided with conventional methods of signal processing. As will be demonstrated, ANNs offer good performance in the presence of both random and correlated noise. They also offer good immunity to variations in amplitude.

4.2 Design of Narrowband Beamformer

The problem of designing a narrowband beamformer to determine the arrival angle from space sampled data can be solved by using ANNs. However this cannot be done with one level of perceptrons. This becomes apparent when the formula for the perceptron is considered. [4]

$$OUTPUT = \frac{1}{1 + e^{-NET}} \qquad (4.1)$$

$$\text{Where } NET = \sum_{i=1}^{N} W_i \cdot S_i$$

Since the received signal, S_i, is a bipolar cyclic signal there will be a time when $S_{i+t} = -S_i$. Therefore, for every +NET result there will also be a -NET result. Since +NET should turn the perceptron on, -NET will definitely turn it off. This can also be seen from a graph of the 2 signal system shown in Figure 20. In this plot the output of a two sensor system is plotted. Three signals are presented. These signals are:

(1) $SENSOR_X = \mathrm{SIN}(\omega t)$; $SENSOR_Y = \mathrm{SIN}(\omega t + \alpha_1)$

(2) $SENSOR_X = \mathrm{SIN}(\omega t)$; $SENSOR_Y = \mathrm{SIN}(\omega t + \alpha_2)$

(3) $SENSOR_X = \mathrm{SIN}(\omega t)$; $SENSOR_Y = \mathrm{SIN}(\omega t + \alpha_3)$

$$\alpha_i = \pi \mathrm{SIN}(\omega_i), \quad \theta_1 = 20°, \quad \theta_2 = 30°, \quad \theta_3 = 40°.$$

The problem for the ANN to learn and process is one of separating each of these signals. It is obvious that these signals cannot be linearly separated. There is no way one straight line can be drawn separating any of the curves into two linearly separable regions.

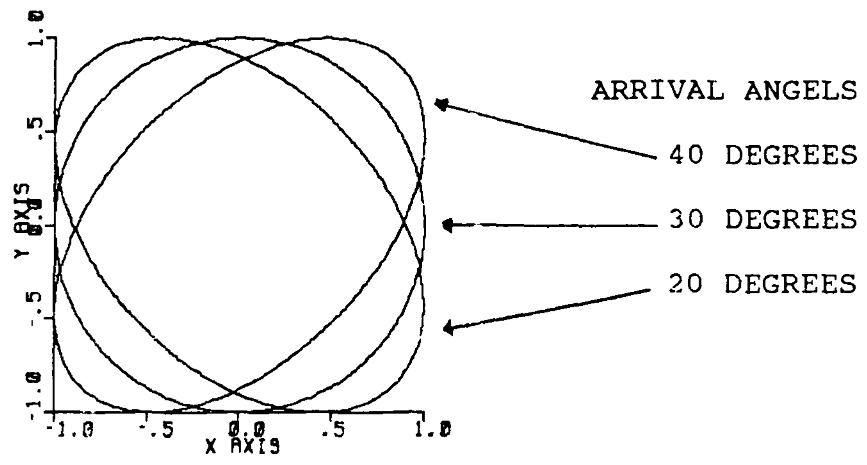


Figure 20

Graph of 2 Signal System

In order to design a narrowband beamformer multiple perceptrons will be required. It is suggested that this can be done by dividing the waveform to be analyzed into segments and then designing each perceptron in the first level of the network to key on one and only one segment of the waveform. The segments which make up the desired response are then OR'ed together with a final perceptron. An example of such a network is shown is Figure 21.

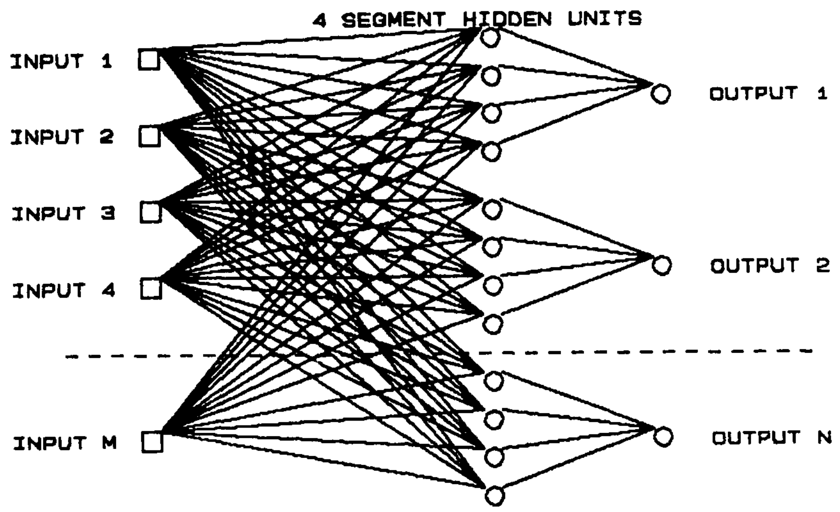


Figure 21

Segmented Network

In this chapter four methods are used to demonstrate how the weights for the network can be found. These methods are:

(1) By Mathematical Analysis.

(2) By training each level of the network separately with the Delta Rule and knowledge of what the output of each level should be.

(3) By Backpropagation training.

(4) By using preprocessing of the sensor inputs and Backpropagation.

4.3 Mathematical Analysis of Beamformer

When the approximate formula of the input signal is known, a mathematical analysis can be performed and values for the weights can be calculated. The basic conditions to be satisfied for each of the arrival angle bands, θ, and time segment, t, are:

(1) OUTPUT = f(NET) = 1 for $\theta_2 > \theta > \theta_1$ and $t_1 > t > t_2$

(2) OUTPUT = f(NET) = 0 for $\theta_2 < \theta$ and for all t

(3) OUTPUT = f(NET) = 0 for $\theta_1 > \theta$ and for all t

(4) OUTPUT = f(NET) = Don't Care Otherwise.

Since the sigmoid function, $f(NET) = \frac{1}{1+e^{-NET}}$, is monotonically increasing and bounded by 0 and 1, it will approach 1 as NET becomes very large and 0 as NET becomes very small.

Therefore for f(NET) to equal 1 the following inequality must be satisfied.

$$NET \gg 0; \quad NET = \sum W_i S_i - THRESHOLD; \quad S_i = B \cdot \mathrm{SIN}(\omega t + i\alpha_i)$$

or

$$\sum W_i \cdot B \cdot \mathrm{SIN}(\omega t + i\alpha_i) \gg THRESHOLD \qquad (4.2)$$

For f(NET) to equal 0, the opposite of the above inequality must be solved.

$$\sum W_i \cdot B \cdot \mathrm{SIN}(\omega t + i\alpha_i) \ll THRESHOLD \qquad (4.3)$$

$$\alpha = \frac{2\pi d\,\mathrm{SIN}(\theta)}{\lambda}$$

Where θ is the arrival angle,

λ is the wave length.

B is the amplitude of the waveform.

d is the distance between sensors.

To minimize the error, the formula

$$V = \sum W_i \cdot B \cdot \mathrm{SIN}\left(wt + i\frac{2\pi d\,\mathrm{SIN}(\theta)}{\lambda}\right) \gg THRESHOLD \qquad (4.4)$$

should be maximized for $\theta_1 < \theta < \theta_2$, and the formula

$$V = \sum_{i=1}^{N} W_i \cdot B \cdot \mathrm{SIN}\left(wt + i\frac{2\pi d\,\mathrm{SIN}(\theta)}{\lambda}\right) \ll THRESHOLD \qquad (4.5)$$

should be minimized for $\theta_2 < \theta < 90°$ and $0° < \theta < \theta_1$ for all t.

The expected values of these formulas are found to be

$$E[V] = \int_{\theta_1}^{\theta_2} f(\theta) \int_{t_1}^{t_2} f(t) \sum_{i=1}^{N} W_i \cdot B \cdot \mathrm{SIN}(wt + i\alpha)\,dt\,d\theta \qquad (4.6)$$

$$\alpha = \frac{(2\pi d)}{\lambda}\mathrm{SIN}(\theta).$$

where $f(\theta)$ and f(t) are probability density functions.

The probability of any arrival angle, θ, or time segment, t, of the sine wave is usually a uniform distribution. The result of the integration yields

$$E[V] = \frac{1}{\omega(t_2 - t_1)} \cdot \frac{1}{\theta_2 - \theta_1} \sum_{i=1}^{N} W_i \int_{\theta_1}^{\theta_2} B \cdot (\mathrm{COS}(wt_2 + i\alpha) - \mathrm{COS}(wt_1 + i\alpha))\,d\theta. \qquad (4.7)$$

This formula can be solved by composite numerical integration. The maximum values for the function $OUTPUT = \sum W_i f(i)$ should occur when they are fully correlated with f(i)=Wi. Therefore the weights are calculated to be equal to the above formula.

To demonstrate the effect of these weights the equation can be tested with the known signal, $A \cdot SIN(\omega t + \alpha i)$. The output of the neuron can be expressed as

$$OUTPUT = f(NET). \tag{4.8}$$

$$where \quad NET = \sum_{i=1}^{N} A \cdot SIN(\omega t + \alpha i) \cdot W_i .$$

Substituting the calculated values for W_i into this formula and writing t_1 and t_2 in terms of the width of the time segment and the center of the time segment yields:

$$OUTPUT = \frac{1}{1 + e^{-NET}} \tag{4.9}$$

$$NET = \frac{-2 \cdot A \cdot B}{\theta_2 - \theta_1} \cdot SINC(\omega t_b) \sum_{i=1}^{N} \cdot SIN(\omega t + i\alpha) \cdot \int_{\theta_1}^{\theta_2} SIN(\omega t_c + i\alpha) d\theta \tag{4.10}$$

$$t_c = \frac{(t_2 + t_1)}{2} \text{ and } t_b = \frac{(t_2 - t_1)}{2}$$

$$\alpha = \frac{2\pi d}{\lambda} SIN(\theta).$$

It can be noted that this formula will provide high values proportional to A, and $SINC(\omega t_b)$. Therefore the more hidden neurons that are used to subdivide the waveform into small time segments, t_b, the higher the neuron's NET output will be.

A plot of this formula's response is shown in Figure 22. In this example the formula is evaluated with θ_1 and θ_2 equal to 20 and 30 degrees respectively. The time segment was centered at zero, (t_c=0). The phase can be calculated from the arrival angle by the formula $\alpha = \frac{2\pi d \sin(\theta)}{\lambda}$. This plot of the perceptron's output as a function of time and arrival angle demonstrates that the neuron fires when the signal is in both the correct angle bin and time segment bin. The plot also demonstrates that when the signal is outside the correct angle bin the result will be zero.

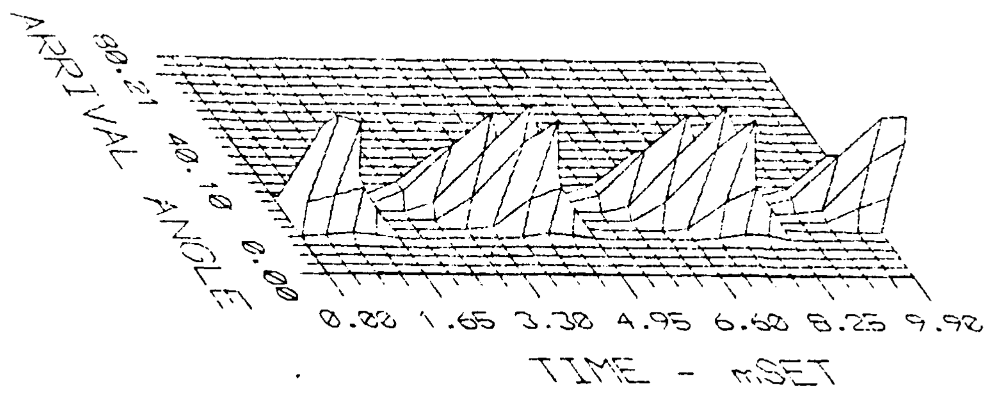


Figure 22

Mathematically Designed Response

One problem should be noted. No matter what angle is being applied to the perceptron, there can be times when the amplitude, A, can be high enough to cause the neuron to excite.

After the outputs of each segment are calculated, they are OR'ed together using one perceptron to provide the desired output. The plot in Figure 22 does not necessarily represent the optimum solution to the problem. It represents only one of many possible solutions and could be one of the local minima into which a backpropagation routine can fall.

4.4 Results of Mathematically Designed Beamformer

Several simulations were made using the weights derived with Equation (4.7). Each simulation set consist of 9 simulations for 9 different arrival angles. These arrival angles were 0-10, 10-20, 20-30, ... 80-90 degrees. The simulations were made using 10 sensor inputs and 2, 3, and 20 hidden units. Plots of these simulations are shown in Figures 23, 24, and 25. Simulations were also run for networks with 4,5 and 10 hidden units. These networks showed very little difference to the ones with 3 and 20 hidden units.

These plots demonstrate that when the number of hidden units is increased to around 4 the network's ability to determine the correct direction is greatly improved. The plots also demonstrate that the lower arrival angles can be identified more accurately than higher arrival angles. A comparison was made between the FFT narrowband beamformer shown is Figure 15 and the mathematically designed beamformer in Figure 25. The percent of correct responses for the FFT beamformer was 97% while the mathematically designed beamformer was only 76%. This was primarily due to the mathematically designed beamformers poor performance for high arrival angles. A comparison itemized by arrival angle is shown in Table 2. It should be noted that the mathematically designed beamformer is designed to detect equal arrival angles but the DFT beamformer detects equal phases.

Arrival Angle	% Correct, DFT Beamformer	% Correct, Math Beamformer (20 Hidden Units)
0 to 10		92.75
0.0 to 3.17	98.33	
10 to 20		86.81
3.98 to 10.38	96.67	
20 to 30		87.36
11.2 to 17.75	97.22	
30 to 40		84.63
18.6 to 25.45	97.22	
40 to 50		71.69
26.3 to 33.66	97.22	
50 to 60		68.42
34.6 to 42.75	96.66	
60 to 70		61.58
43.7 to 53.38	96.11	
70 to 80		63.92
54.4 to 67.59	96.67	
80 to 90		66.83
67.6 to 90.	99.44	

Table 2

Comparison between Beamformers

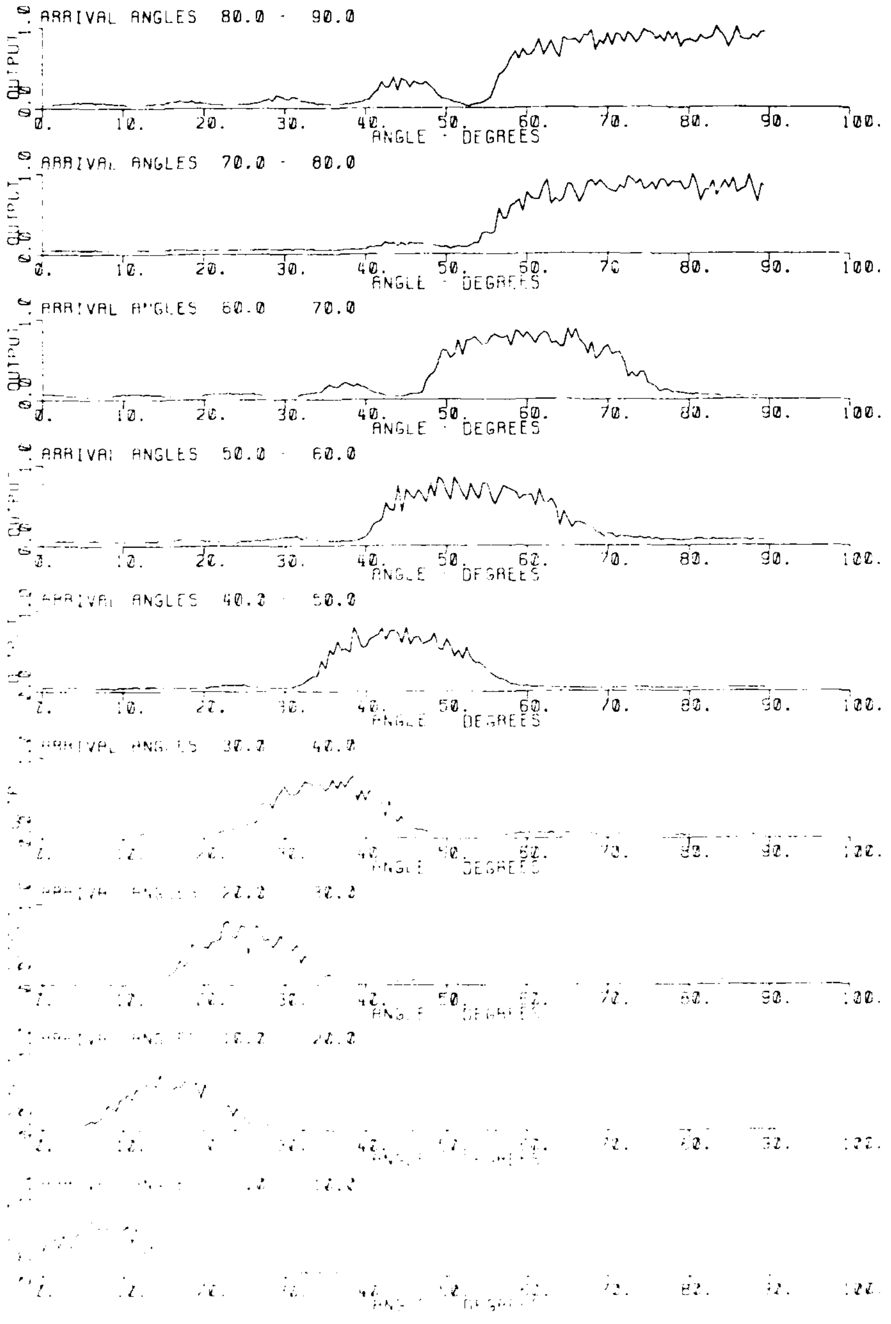


Figure 23

Mathematically Designed ANN Using 2 Hidden Units

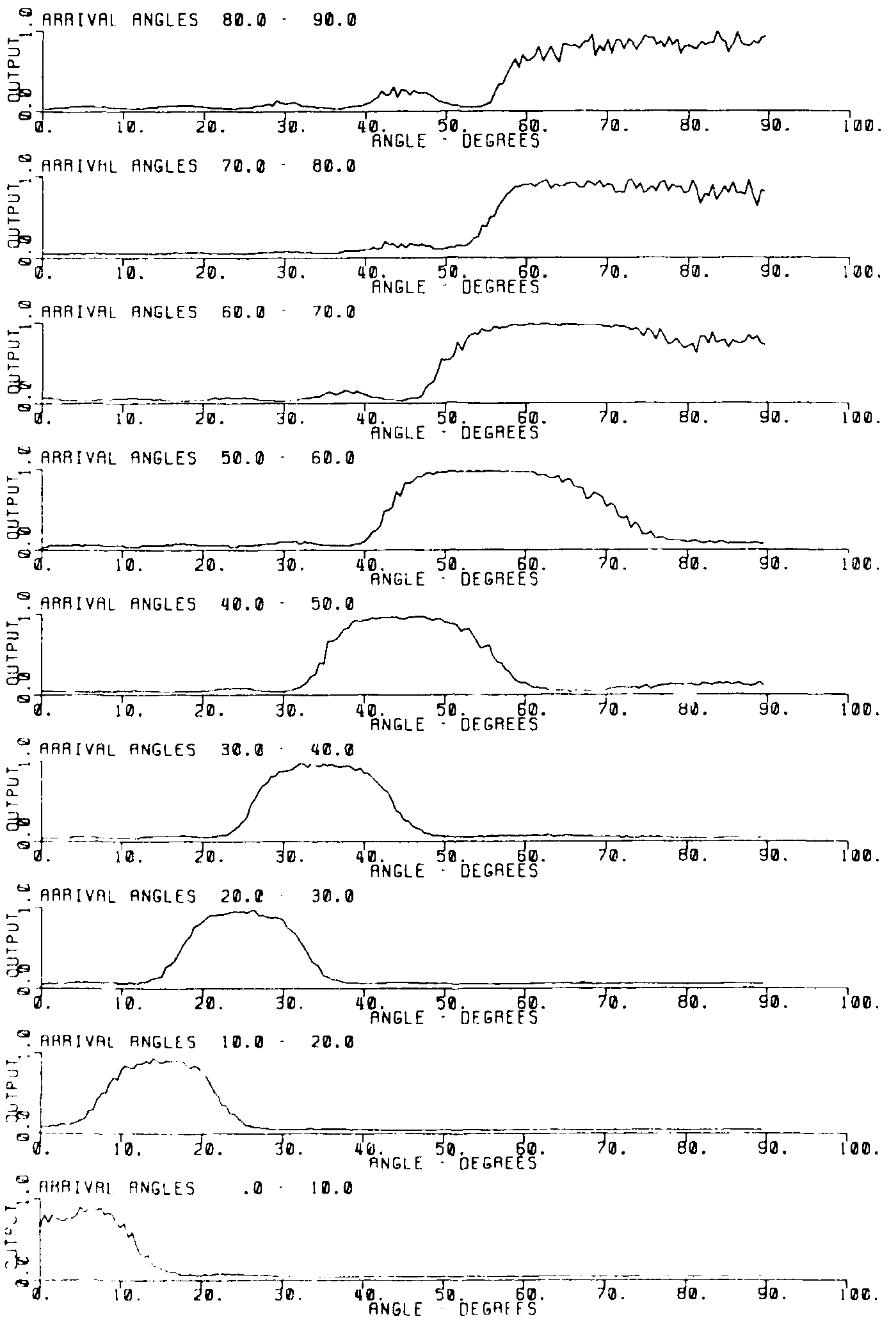


Figure 24

Mathematically Designed ANN Using 3 Hidden Units

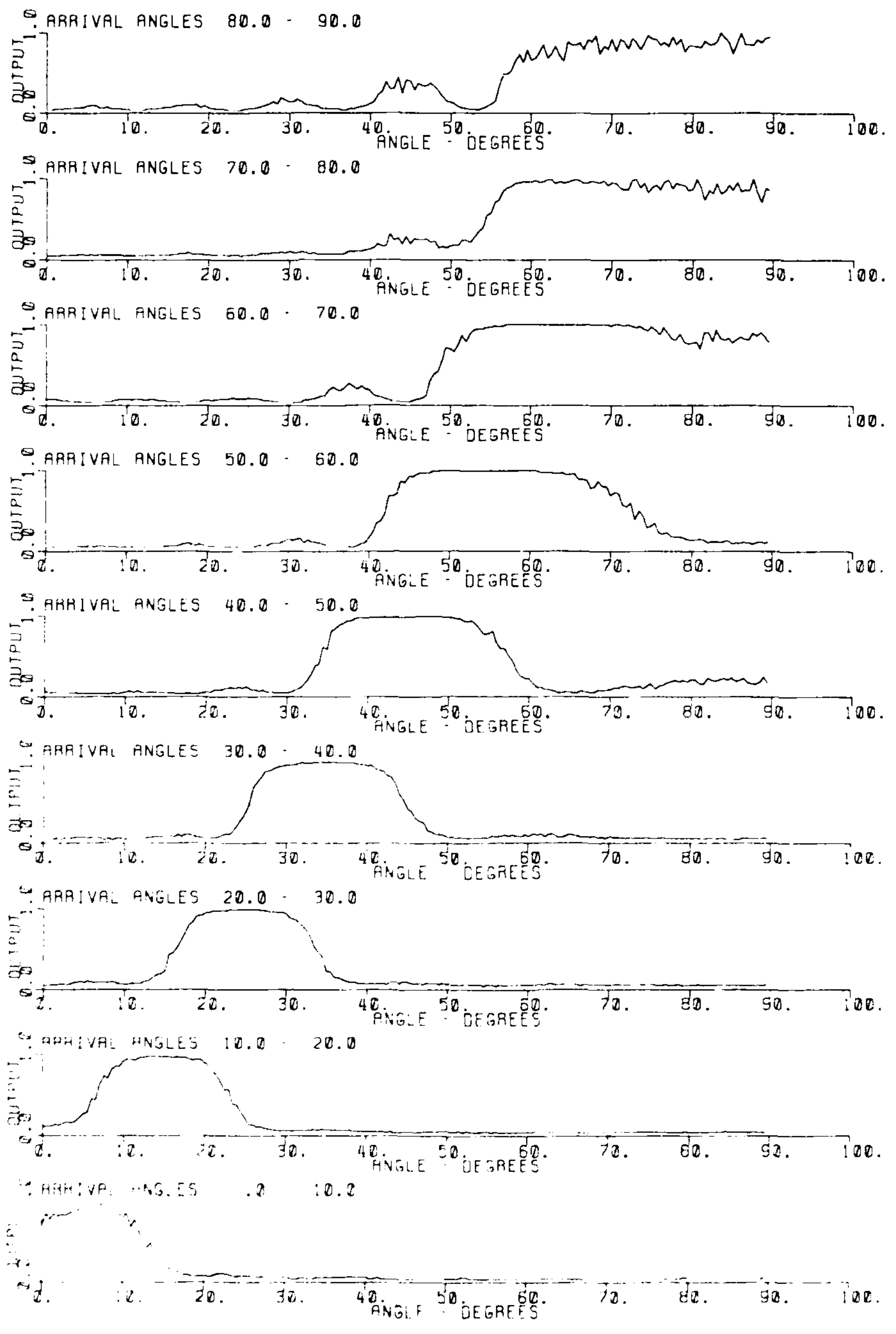


Figure 25

Mathematically Designed ANN Using 20 Hidden Units

4.5 Delta Rule Training of Narrowband Beamformer

The first training method that will be used to train the narrowband beamformer is the Delta Rule. When using the Delta Rule each level is trained independently based on knowledge of what that level's output should be. The network shown in Figure 26 is used.

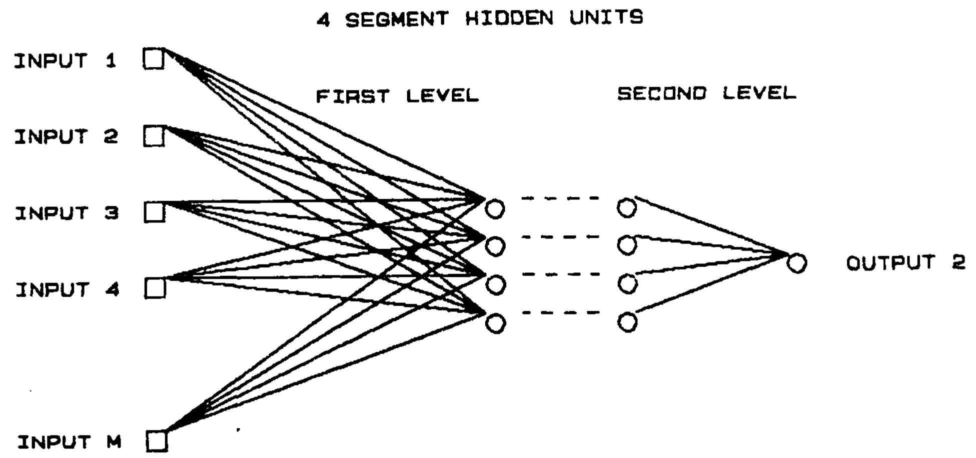


Figure 26

Narrowband Beamformer for One Arrival Angle

Many neural network models are divided into two main sections. The first section performs "feature extraction" to separate the data's important characteristics needed by the second level. The second level is the network that learns to process these features and learns the correct response to different stimuli. The networks in Figure 26 can be considered to be such a system. The first level extracts the features which are the different time segments of the angle to be detected and the second level is trained to give the proper response.

To implement this model each arrival angle band is trained separately. Every input sensor is connected to every hidden level neuron. Each hidden level neuron is designed to respond to the presence of one segment of one arrival angle band. The outputs of the hidden units within a segment are then combined in the second level to provide an output that represents the presence of a signal within the arrival band at any time. The following procedure is used to train each of the networks.

(1) Equation (4.7) is used to calculate the weights for each segment of the first level.

(2) The last level is trained using the Delta Rule to provide an optimum set of weights for the final level.

This method is a very practical method of training the network. When the output of the hidden units can be determined by training or analysis, the Delta Rule will provide a fast method of calculating optimum weights for the final level.

4.6 Results of Training with the Delta Rule.

The network designed in the previous section was used to simulate the same three configurations presented in the previous section. The first level was trained using Equation (4.7). The network was implemented using 2, 3, and 20 hidden units. Simulations using 5 and 10 hidden units were also implemented, but they were almost identical to the simulations conducted with 3 and 20 hidden units. The final level was trained with the Delta Rule. A learning rate between .1 and .3 and a momentum term of .9 were used in the learning algorithm. These values were chosen by trying various sets and noticing which ones caused the network to converge faster. The rate of convergence and the minima, local or global, to which the network converges is a function of the input patterns used to train the network and the random values to which the weights are initially set. Since the values are not deterministic, they must be approximated from experimental results. Adjustments to the weights were made after every pattern was presented to the network.

The results of these three simulations are shown in Figures 27 through 29. These plots indicate that when the final perceptrons are trained to find optimum weights for this last level the results will be improved. These plots are still far from ideal. The highest angles are still very erroneous and the transition bands are not sharp. When the weights for the first level of perceptrons were calculated, it was assumed that each element should represent an equal time segment. This was not a good assumption. It would be desirable to allow the first level of perceptrons to train also so they could find values that were more near the optimum level. A comparison between the mathematically designed beamformer and the Delta

Rule trained beamformer is shown in Table 3. This comparison indicates that training can provide better results than the mathematical method used previously. Comparing Table 3 with Table 2 indicates that neither the Delta Rule training method nor the mathematical method are nearly as accurate as the DFT approach. It can be noted that the DFT beamformer's worst arrival angle gave 96.11 % correct response while the mathematical and Delta Rule beamformers' best arrival angle gave only 92.75 % and 93.31 % correct results respectively.

Arrival Angle	Math Designed Beamformer (20 Hidden Units)	Delta Rule Beamformer (20 Hidden Units)
0 - 10	92.75	91.64
10 - 20	86.81	92.69
20 - 30	87.36	93.28
30 - 40	84.63	93.31
40 - 50	71.69	90.61
50 - 60	68.42	85.58
60 - 70	61.58	72.06
70 - 80	63.92	79.97
80 - 90	66.83	72.28

Table 3

Comparison of Delta Rule Beamformer

4.7 Backpropagation Training of Beamformer

Backpropagation is the preferred method for training such a network. With backpropagation no mathematical analysis or prior knowledge about the hidden level's outputs is required. To demonstrate this backpropagation was used to train a network

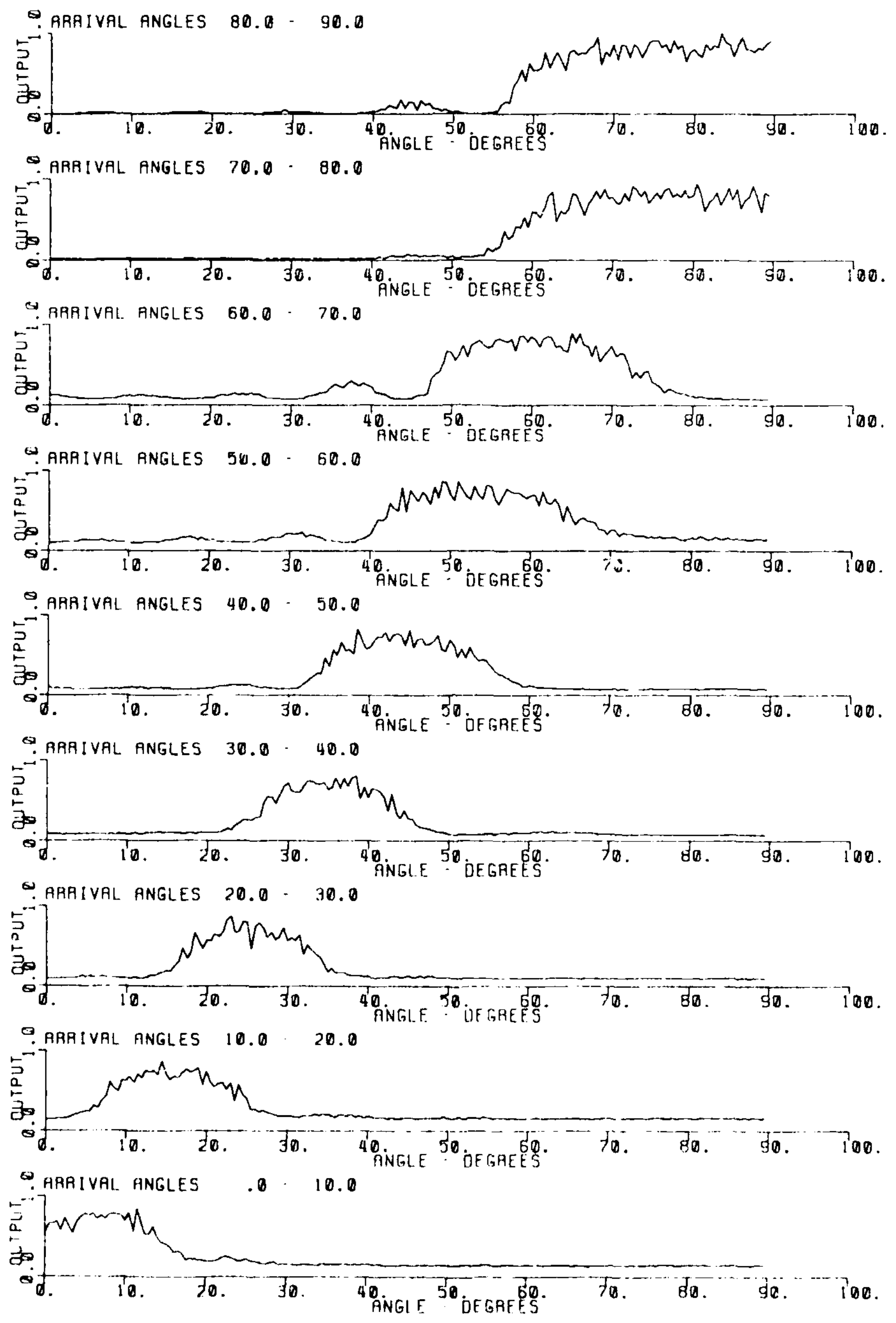


Figure 27

Delta Rule Trained ANN with 2 Hidden Units

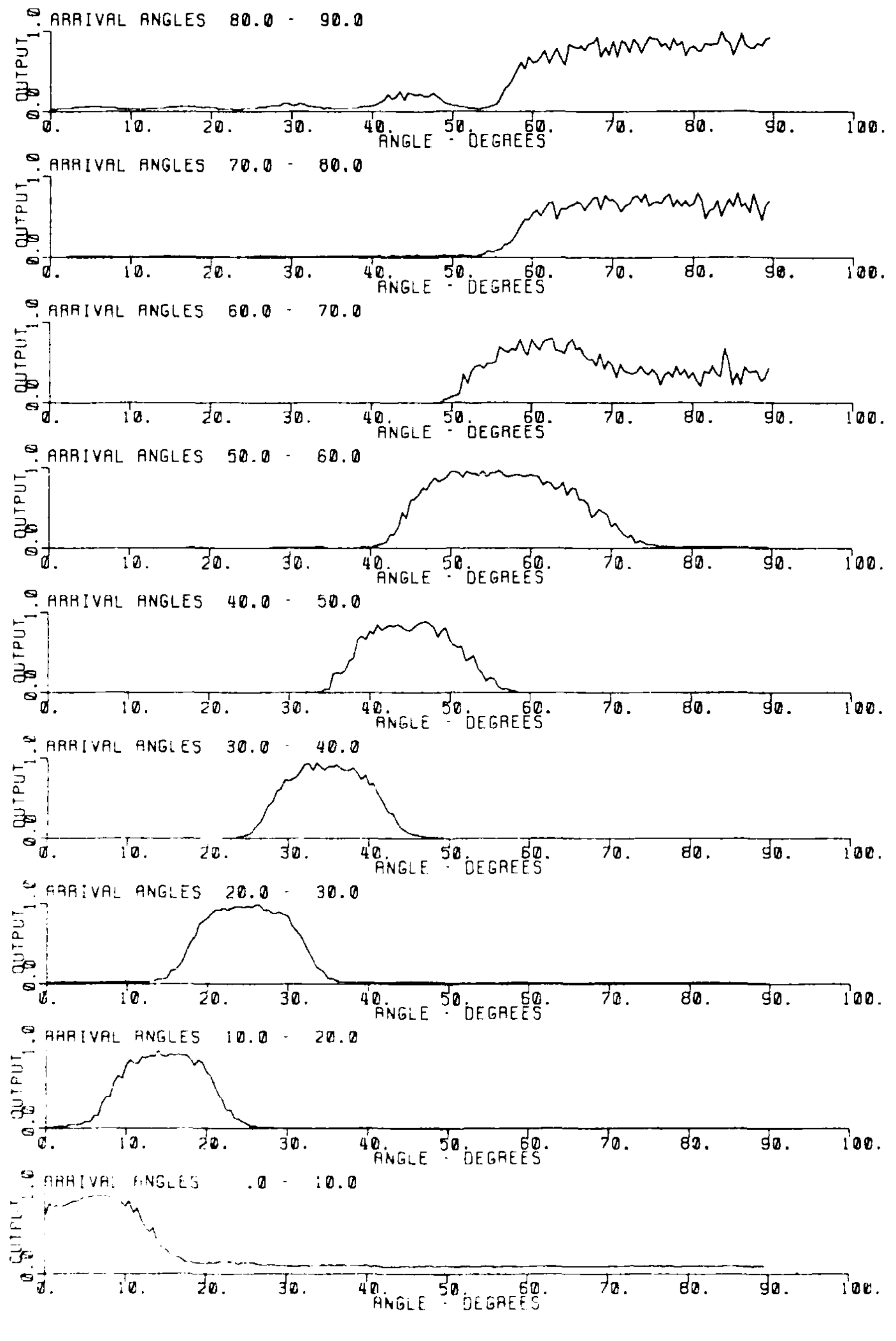


Figure 28

Delta Rule Trained ANN with 3 Hidden Units

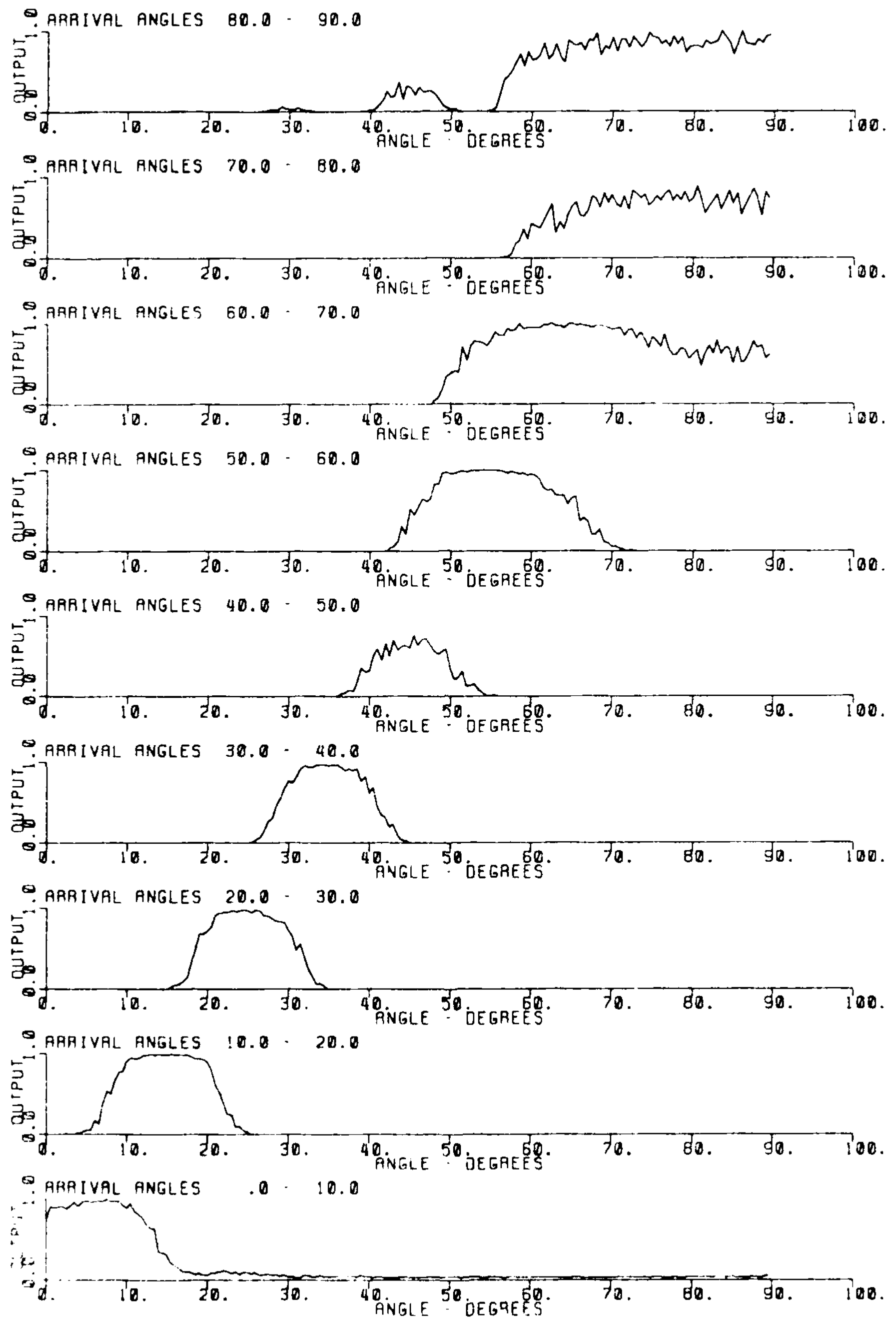


Figure 29

Delta Rule Trained ANN with 20 Hidden Units

comparable to Figure 26. In this network two levels of neurons were used as shown in the network of Figure 30.

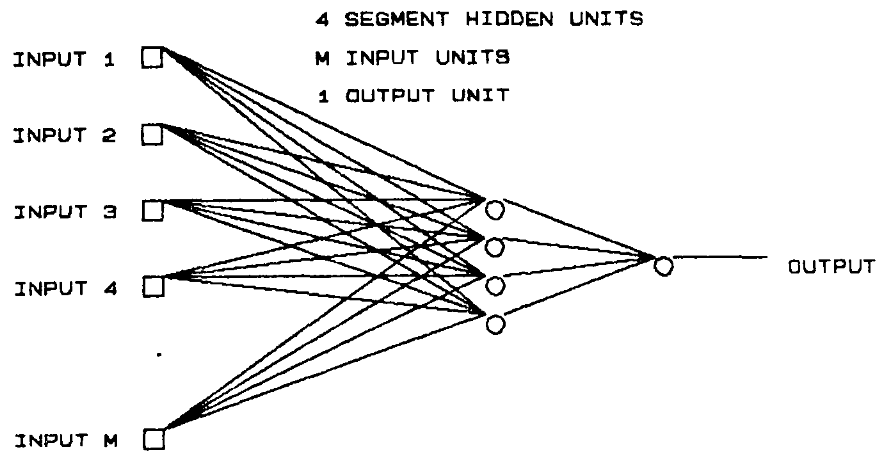


Figure 30

Narrowband Beamformer

Two levels were used since it is known that any convex pattern can be separated from its background in two levels. Backpropagation can be especially important for several reasons. The input signal is not always one for which a simple algebraic formula is known. There can also be nonlinearities in the media through which the data propagates. Mechanical or electrical noise can also add complexity to the problem. Problems emanating from sensor coupling, calibration, linearity, or placement can also complicate a formal analysis of the problem. When the problem is very difficult, the use of a mathematical analysis for determining weights is arduous.

However when data from known sources are used to train the network, it will learn the effect of the nonlinearities and adjust the weights accordingly.

4.8 Results of Training with Backpropagation

Backpropagation was used to train the same network used in the Delta Rule Training section and the Mathematical Simulation Section. Again 2, 3, and 20 hidden units were used. In these simulations the average value of the network output is plotted. This average is taken over an entire cycle. This is accomplished by evaluating the network output at 20 random times and computing the average. These results also demonstrate that performance is improved when more hidden units are used. They also indicate again that the response at higher arrival angles is poor. Since the phase is related to the arrival angle by the formula $\alpha = \frac{2\pi d \operatorname{SIN}(\theta)}{\lambda}$ the phase difference will be much smaller for higher angles bands that lower ones. This would require a much sharper separation for the higher angles. The results of these simulations are shown in Figure 31 through 33. Since the highest two arrival angles are the most difficult to train, an inaccurate response was learned in the 70 to 80 degree angle bin. The results of these simulations are shown in Figure 31 through 33.

A comparison of the backpropagation trained network, the Delta Rule trained network, and the mathematically trained network is shown in Table 4. This comparison shows that the backpropagation method and the DFT method are very similar when used as a narrowband beamformer. The DFT beamformer outperformed the backpropagation beamformer is 3 of the 9 arrival angle bins. It shoud be noted that the DFT beamformer's arrival angle bins were approximately 5 degrees wide compared to 10 degrees for the backpropagation beamformer.

Arrival Angle	Math System	Delta Rule System	Backpropagation System	DFT System
0 - 10	92.75	91.64	99.22	98.33 beam 1
10 - 20	86.81	92.69	98.33	96.67 beam 2
20 - 30	87.36	93.28	98.39	97.22 beam 3
30 - 40	84.63	93.31	98.25	97.22 beam 4
40 - 50	71.69	90.61	97.69	97.22 beam 5
50 - 60	68.42	85.58	97.00	96.66 beam 6
60 - 70	61.58	72.06	94.44	96.11 beam 7
70 - 80	63.92	79.97	90.42	96.67 beam 8
80 - 90	66.83	72.28	97.11	99.22 beam 9

Table 4

Comparison with Backpropagation Beamformer

Simulations were also made in which wider ranges of amplitudes were used. In these simulations backpropagation was used to train the same three networks used in the previous simulations. Amplitude ranges of .5 to 2.0 were used to test the network more stringently. The plots of these simulations are shown

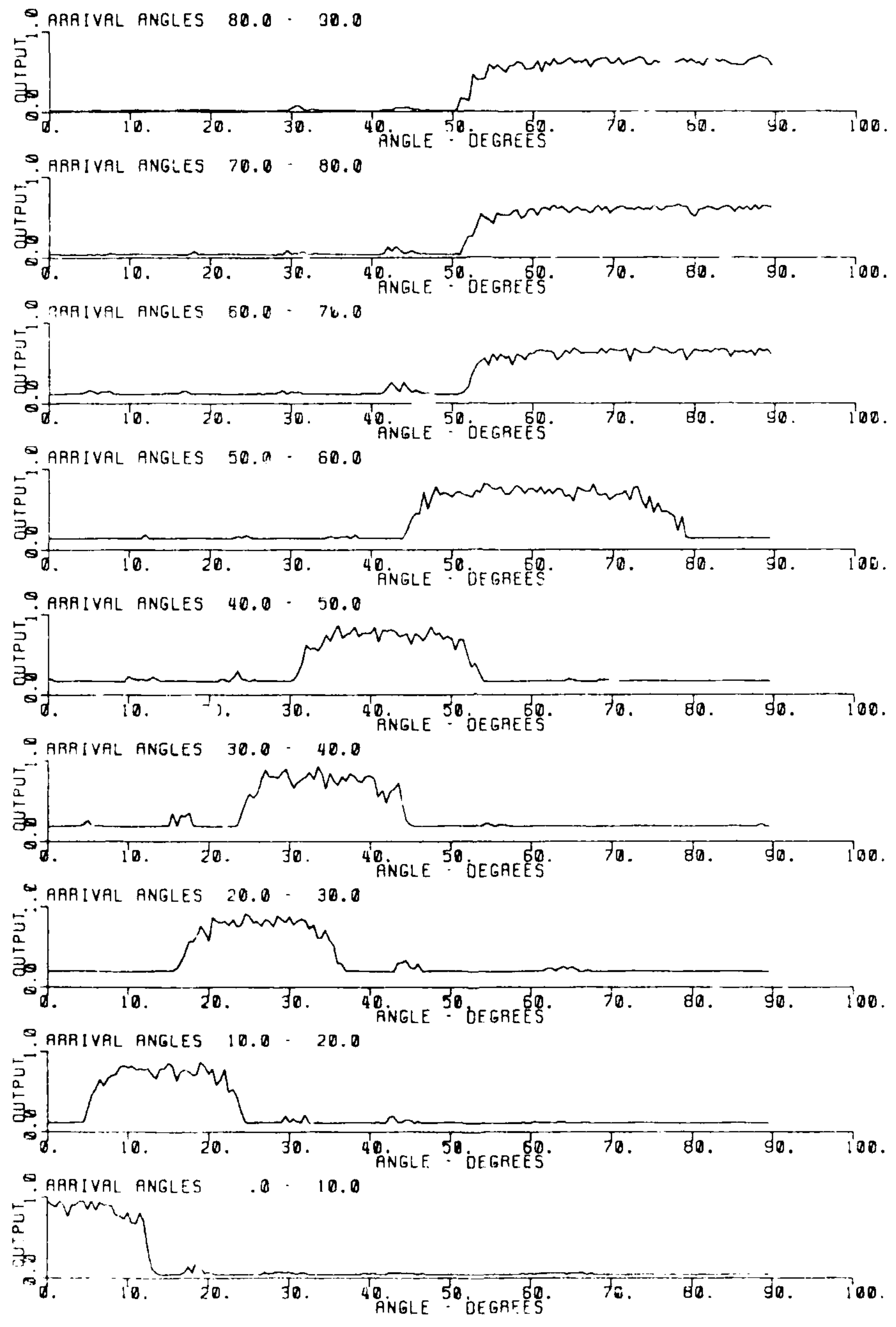


Figure 31

Average Output of Backpropagation ANN with 2 Hidden Units

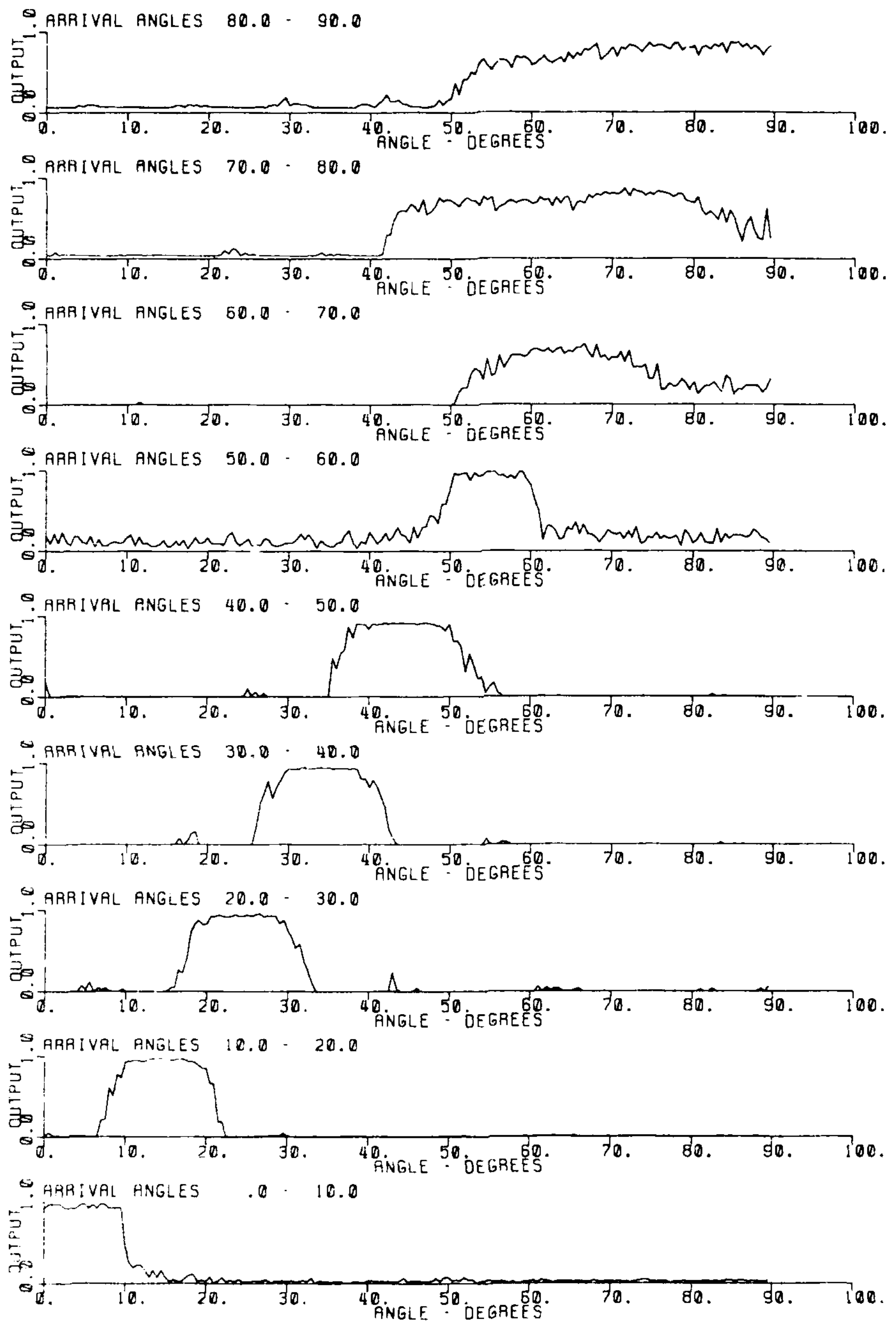


Figure 32

Average Output of Backpropagation ANN with 3 Hidden Units

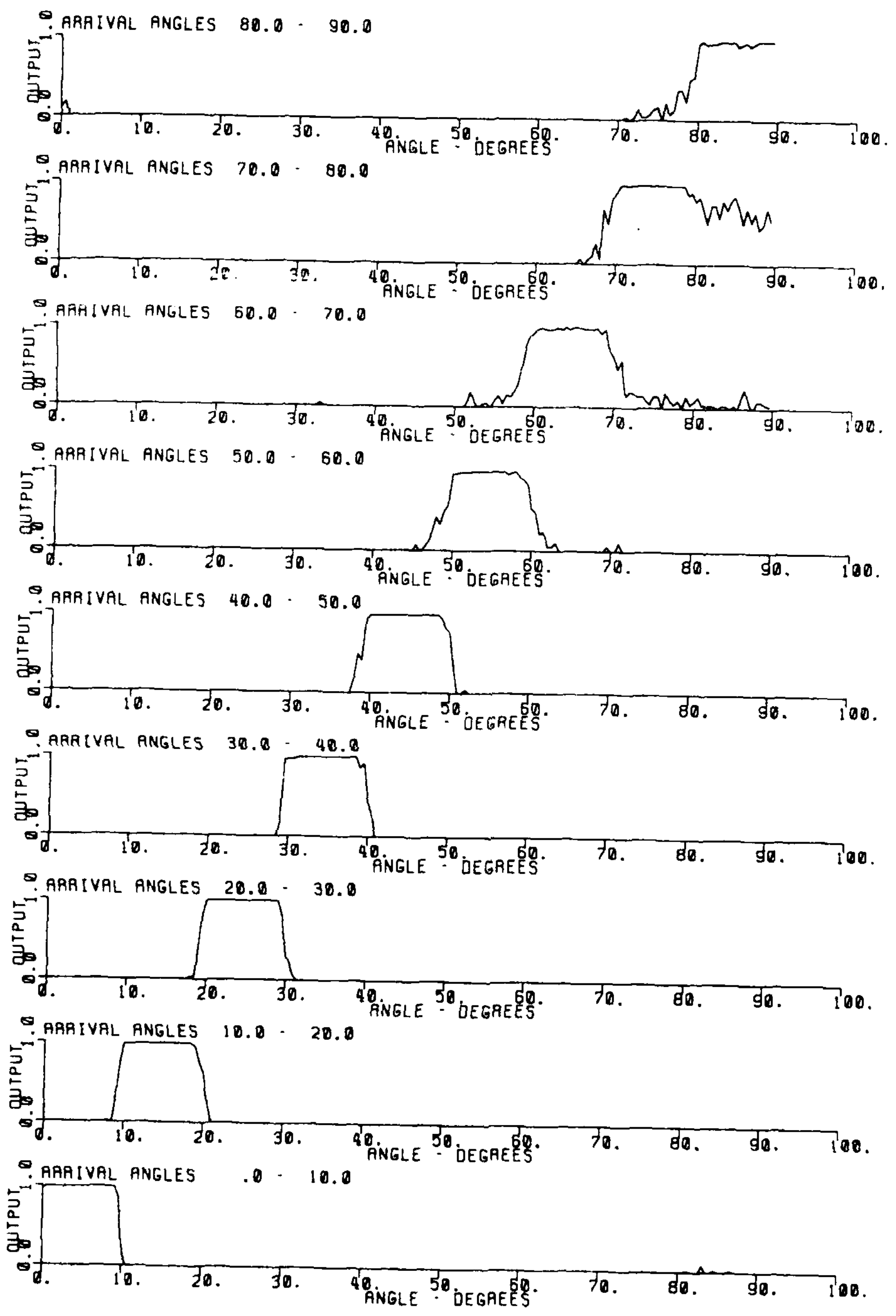


Figure 33

Average Output of Backpropagation ANN with 20 Hidden Units

in Figure 34 through 36. These plots indicate that a network will perform better, if an automatic gain control is used to restrict its amplitude to a narrow range.

The network was also tested for its sensitivity to noise. The peak to peak noise level was set for .2, .4, and .8 on the simulations shown in Figure 37 through 39. These networks used 5 hidden units and an amplitude range of .5 to 1.0. This amplitude choice was taken from a uniform distribution. The uncorrelated noise level was also taken from a uniform distribution. By using the formula $SNR = 10 \cdot LOG_{10} \frac{\sigma^2_{SIGNAL}}{\sigma^2_{NOISE}}$ these noise levels will correspond to a SNR of 19.42 db, 13.34 db, and 7.38 db respectively. These three simulations provided results that were 93.46%, 91.03%, and 81.52% correct. These results show a very serious degradation in response as the signal to noise is decreased from 13.34 db to 7.38 db.

DFT beamformers also degrade as the SNR is decreased. Since the DFT beamformer is an analog system, its performance can be measured by comparing the input SNR to the output SNR. Both the DFT beamformer and conventional beamformers have an array gain improvement on the order of 10log(N), where N is the number of sensors. [5]

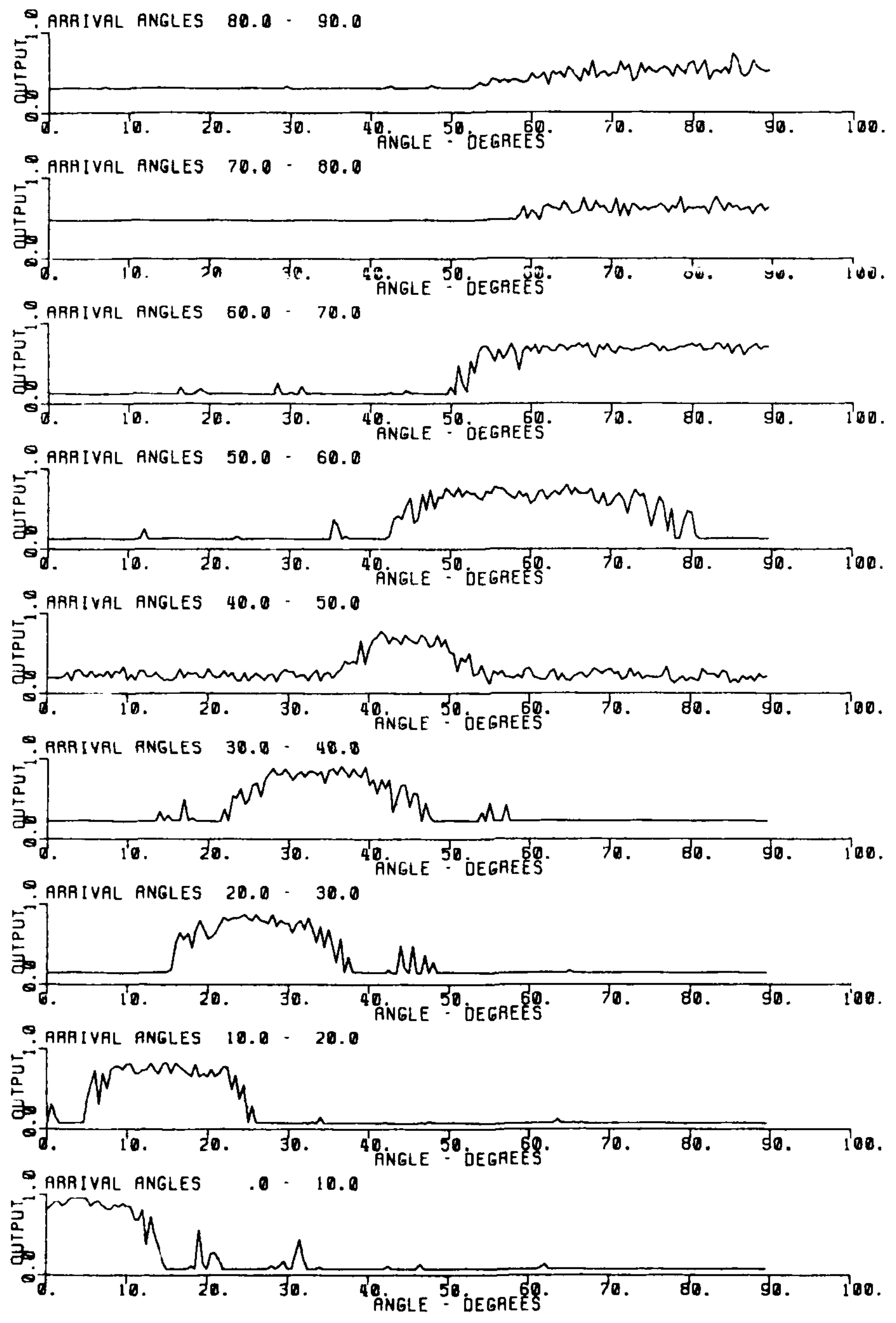


Figure 34

ANN with 2 Hidden Units and Amplitude Range of .5-2.

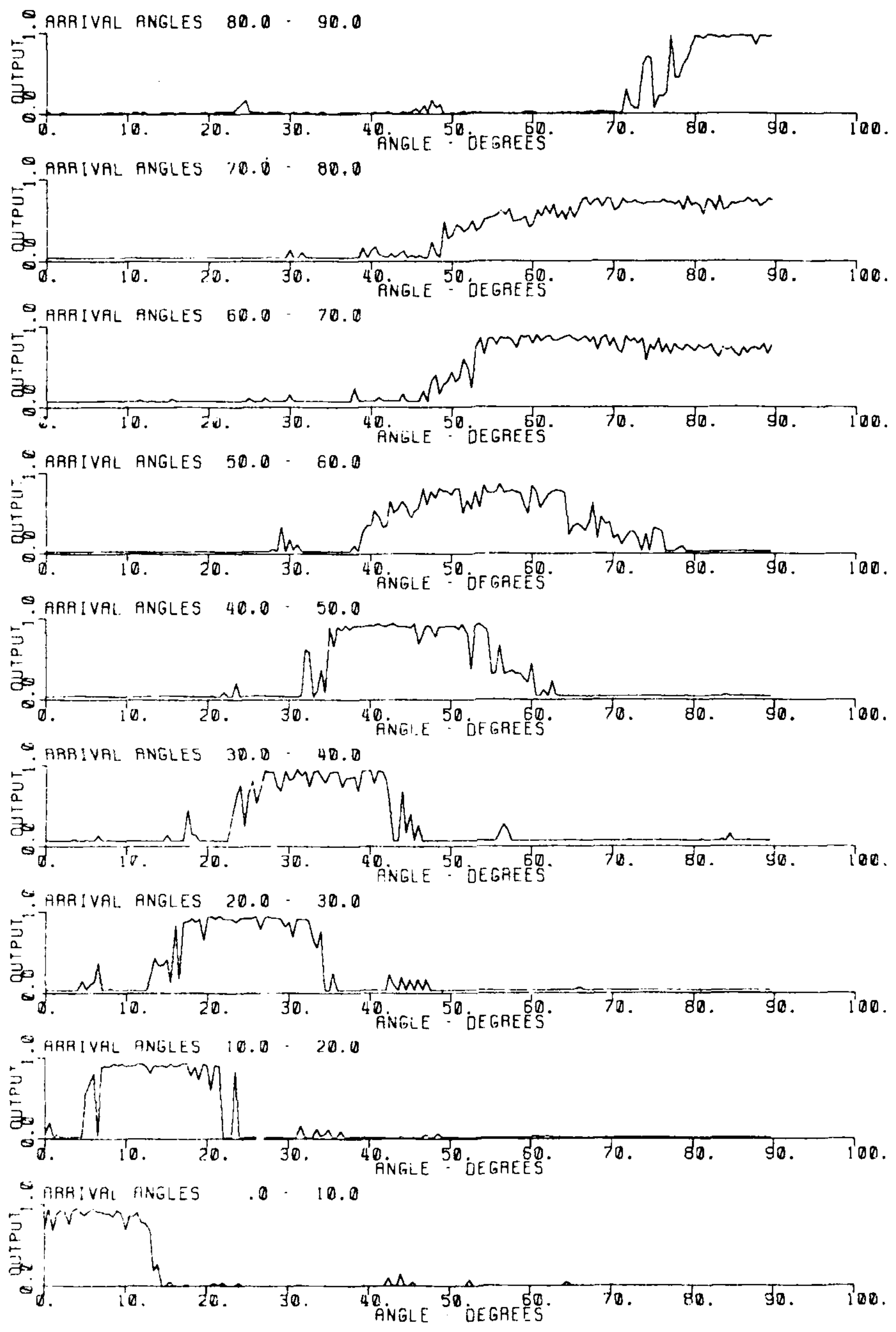


Figure 35

ANN with 3 Hidden Units and Amplitude Range of .5-2.

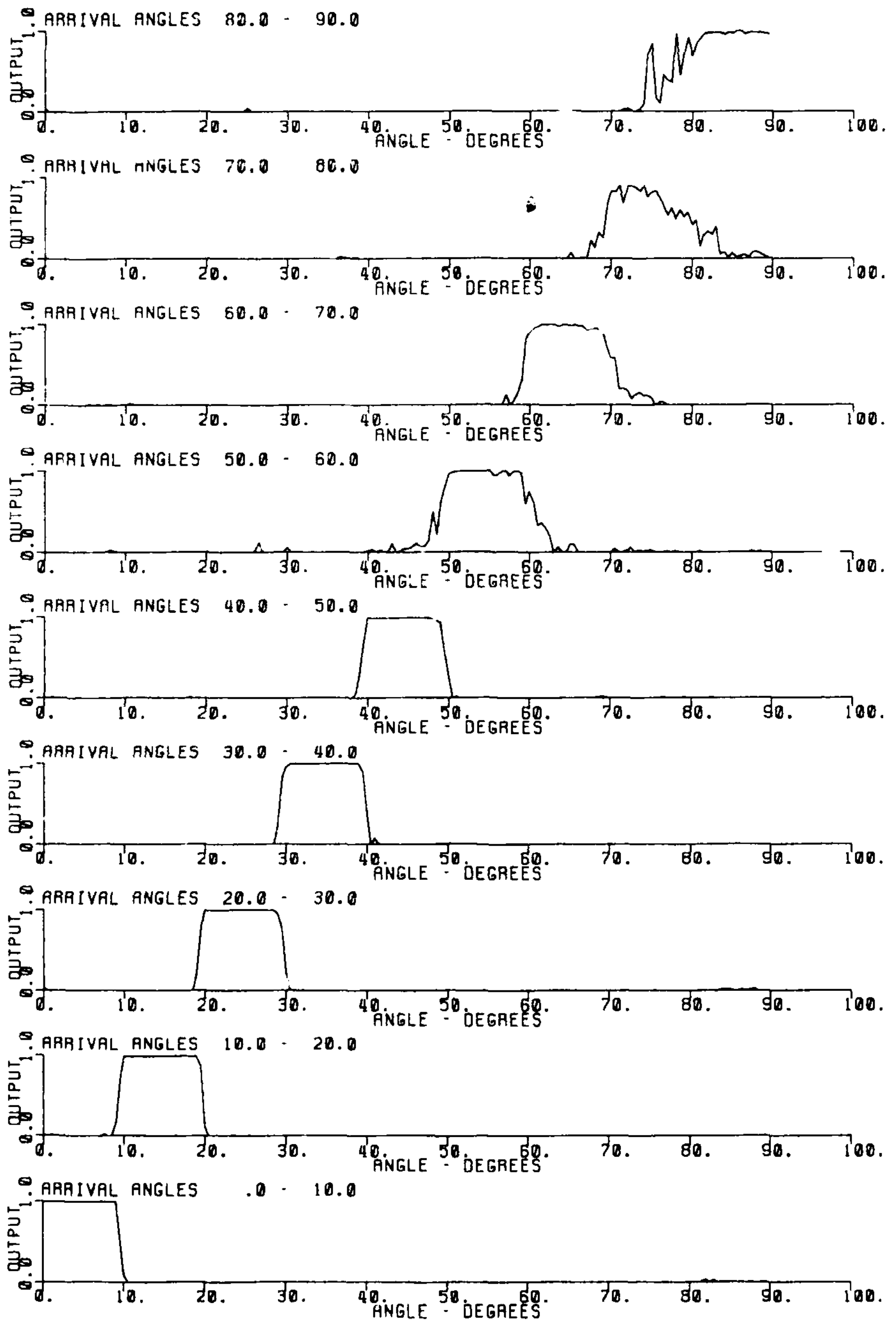


Figure 36

ANN with 20 Hidden Units and Amplitude Range of .5-2.

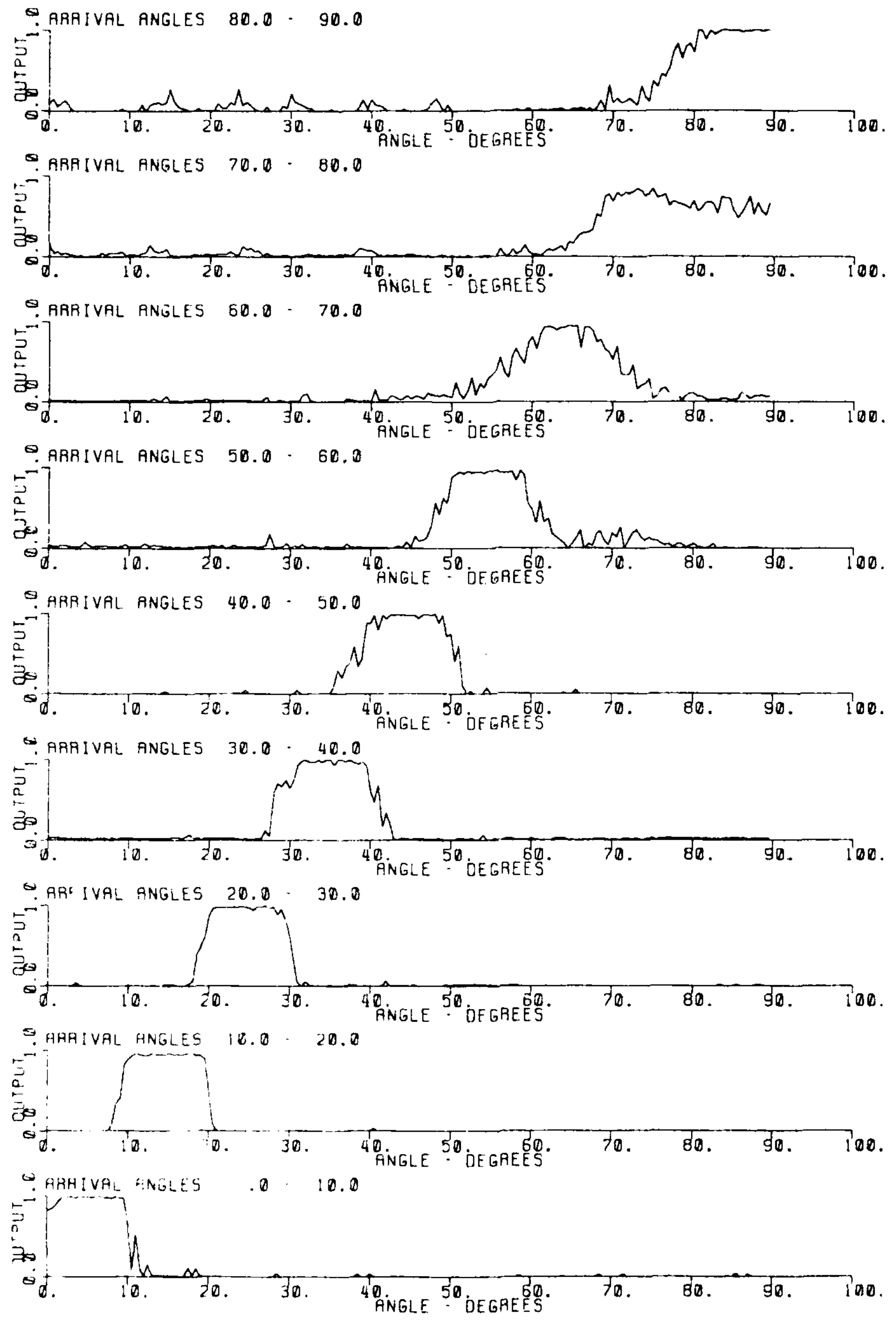


Figure 37

ANN with 2 Hidden Units and Noise of .2

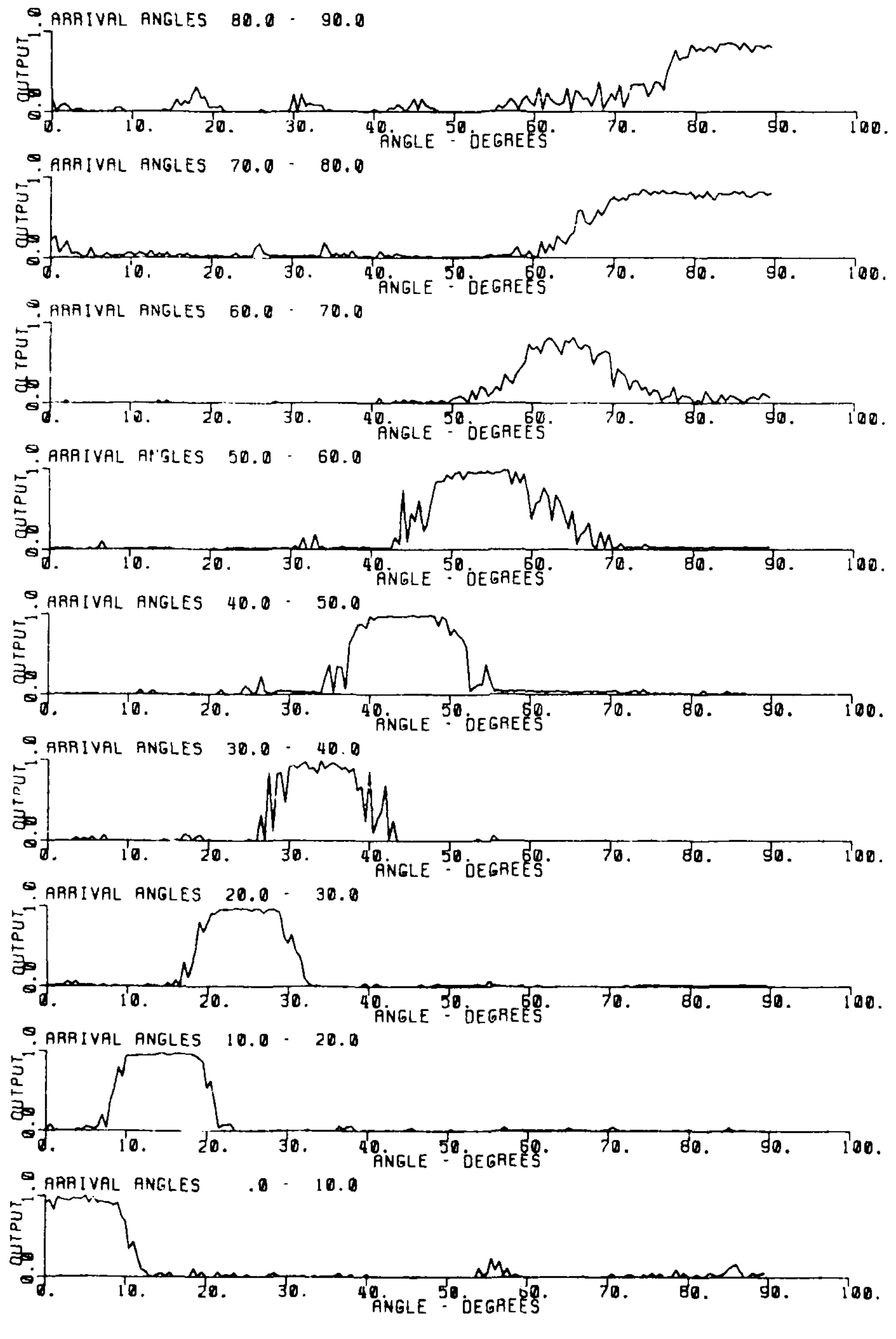


Figure 38

ANN with 3 Hidden Units and Noise of .4

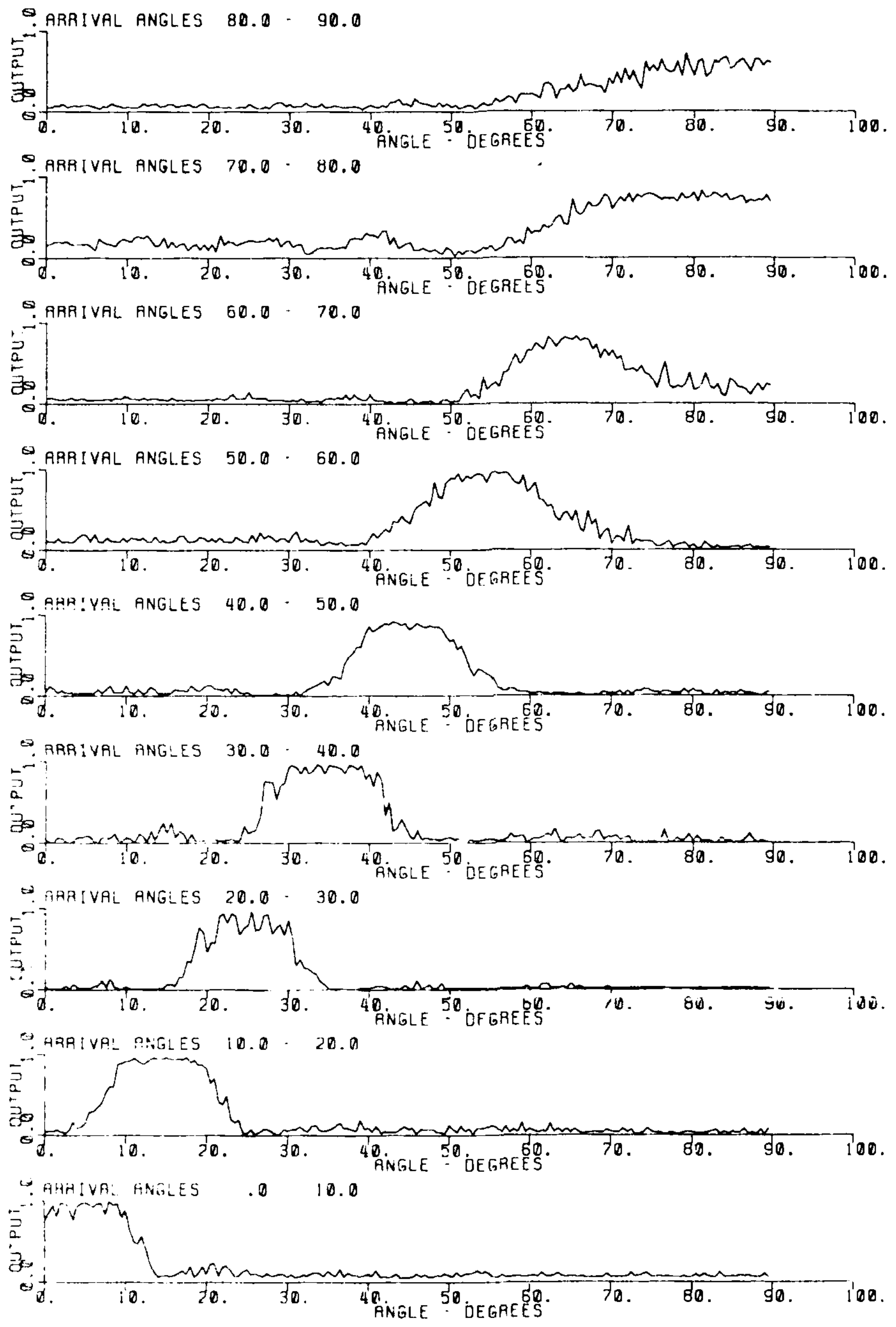


Figure 39

ANN with 20 Hidden Units and Noise of .8

4.9 Narrowband Beamformers with Nonlinear Inputs

Another method of processing narrowband ANN beamformers can be derived from the beamformer's input formula

$$V_i = A \cdot SIN(ut + \alpha i) \qquad (4.11)$$

and

$$\alpha = \frac{2\pi \cdot d \cdot SIN(\theta)}{\lambda}.$$

Let $\alpha_0 = 0$ and the formula for the first two sensors of a system will be

$$V_0 = A \cdot SIN(ut) \qquad (4.12)$$

and

$$V_1 = A \cdot SIN(ut + \alpha_1) = A \cdot SIN(ut)COS(\alpha_1) + A \cdot COS(ut)SIN(\alpha_1).$$

These two equations can then be combined to form one equation independent of time. This equation is

$$V_1^2 - 2 \cdot V_0 V_1 COS(\alpha) + V_0^2 - A^2 SIN^2(\alpha) = 0. \qquad (4.13)$$

If this equation is compiled for each sensor and the sensor next to it, they can be summed into the following form

$$\sum_{i=1}^{N} 2 \cdot V_i - 2 \cdot COS(\alpha) \sum_{i=1}^{N} V_i V_{i-1} - N \cdot A^2 \cdot SIN^2(\alpha) = 0 \qquad (4.14)$$

OR

$$\sum_{i=1}^{N} (W_{1,i} \cdot V_i^2 + W_{2,i} V_i V_{i+1}) + T = 0 \qquad (4.15)$$

which is the form used by the perceptron model if V_i^2 and $V_i V_{i-1}$ are available as inputs.

If preprocessing is used to provide a neural network with both the square of the input and the product of the inputs, the network can learn the weights $W_{1,1}$, $W_{2,1}$, and T. By using only one neuron for each arrival direction the same separations that were made using two levels of neurons and many hidden units can be made. When this network is implemented, it is desirable to separate the arrival angles that are greater than one angle and less than another from all others. An example of such a separation is shown in Figure 40 in which a system with the two inputs X and Y are used.

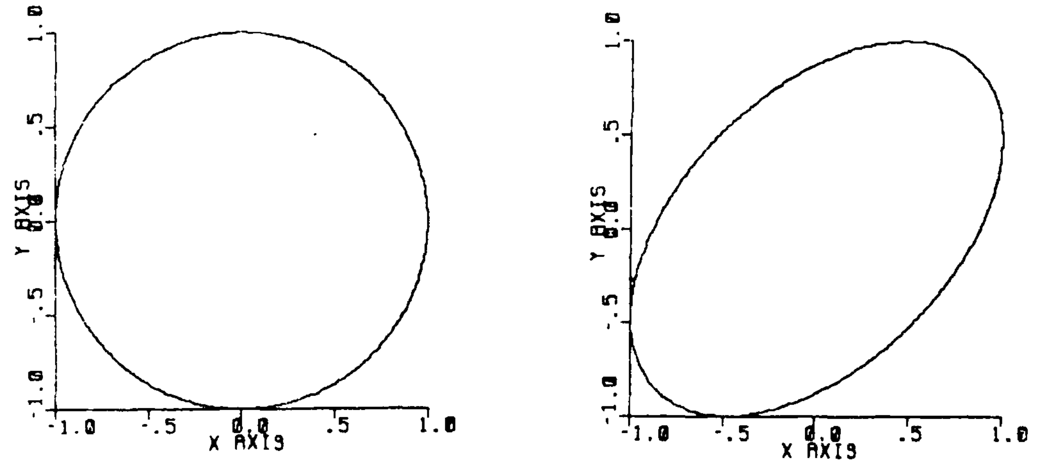


Figure 40

Separation of Angles

However if the amplitude is allowed to vary, the points of intersection between the desired bands and all others increase. Therefore additional hidden units to separate amplitude above or below specified amplitude thresholds are required. Figure 41 shows an example.

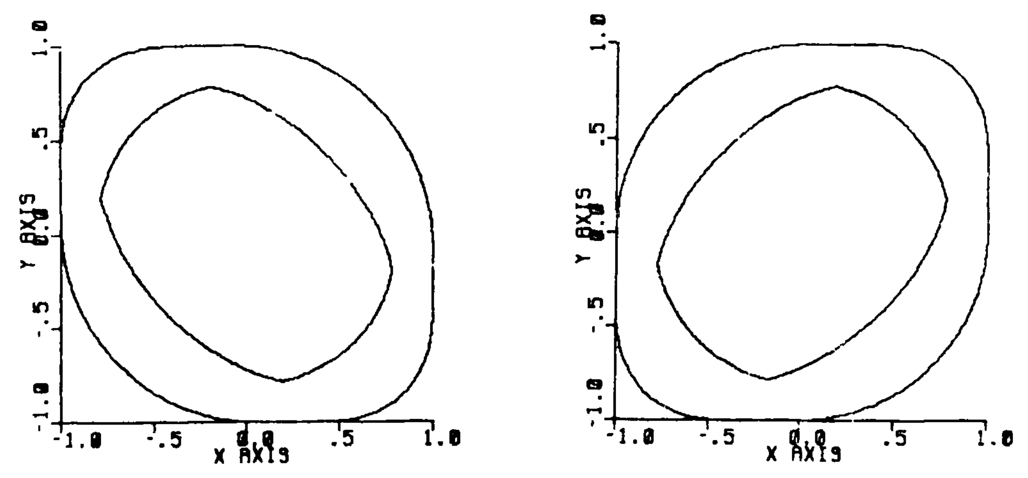


Figure 41

Separation of Angles and Amplitudes

4.10 Results of Beamformer with Nonlinear Inputs

To define the area shown in Figure 41 four neurons can be used to define points within the two curves. An example of the results made with 4 hidden units and 10, and 20 inputs and amplitudes ranges of 0.5 to 1.0 and .5 to 2.0 are shown in Figures 42 through 45. In these simulations that use 20 input sensors a two dimensional beamformer is used. This beamformer had 10 sensors in one line and 10 sensors in a line perpendicular to it.

This method provides superior results to the methods which use only linear inputs. It also requires fewer neurons. Multiplying and squaring inputs is a reasonable modification to make, since the multiplication function is required when ANNs are implemented. Those simulations that are derived from the exact formula for the input will be referred to as "exact" solutions.

A compromise between these two methods can also be demonstrated. By using linear inputs and squared inputs elliptical patterns which closely resemble the exact solution can be realized. An example of the same four trials tested with the exact solution are tested with the elliptical network in Figures 46 through 49. These simulations will be referred to as "elliptical" solutions. No noise was used in either the elliptical or the exact simulations.

4.11 The Network's Dependency on Time

It was shown in Equation (4.7) that the networks using linear inputs are dependent on time and must rely on subdividing the network into segments or output averaging to provide acceptable results. The networks with nonlinear inputs described by Equation (4.15) demonstrate that these networks can be independent of time. However even though the network can be expressed in a formula which is independent of time, there is no guarantee that the solution learned by the network will be completely independent of time. To demonstrate the effect that time has on these networks two simulations are presented using both linear and nonlinear inputs. Figures 50 and 51 present the same two simulations as were shown in Figures 36 and 42. These plots demonstrate that when the linear input systems are used output averaging must be employed to improve the systems response. Figure 50 represents one of the best linear networks and Figure 51 represents one of worst exact simulations.

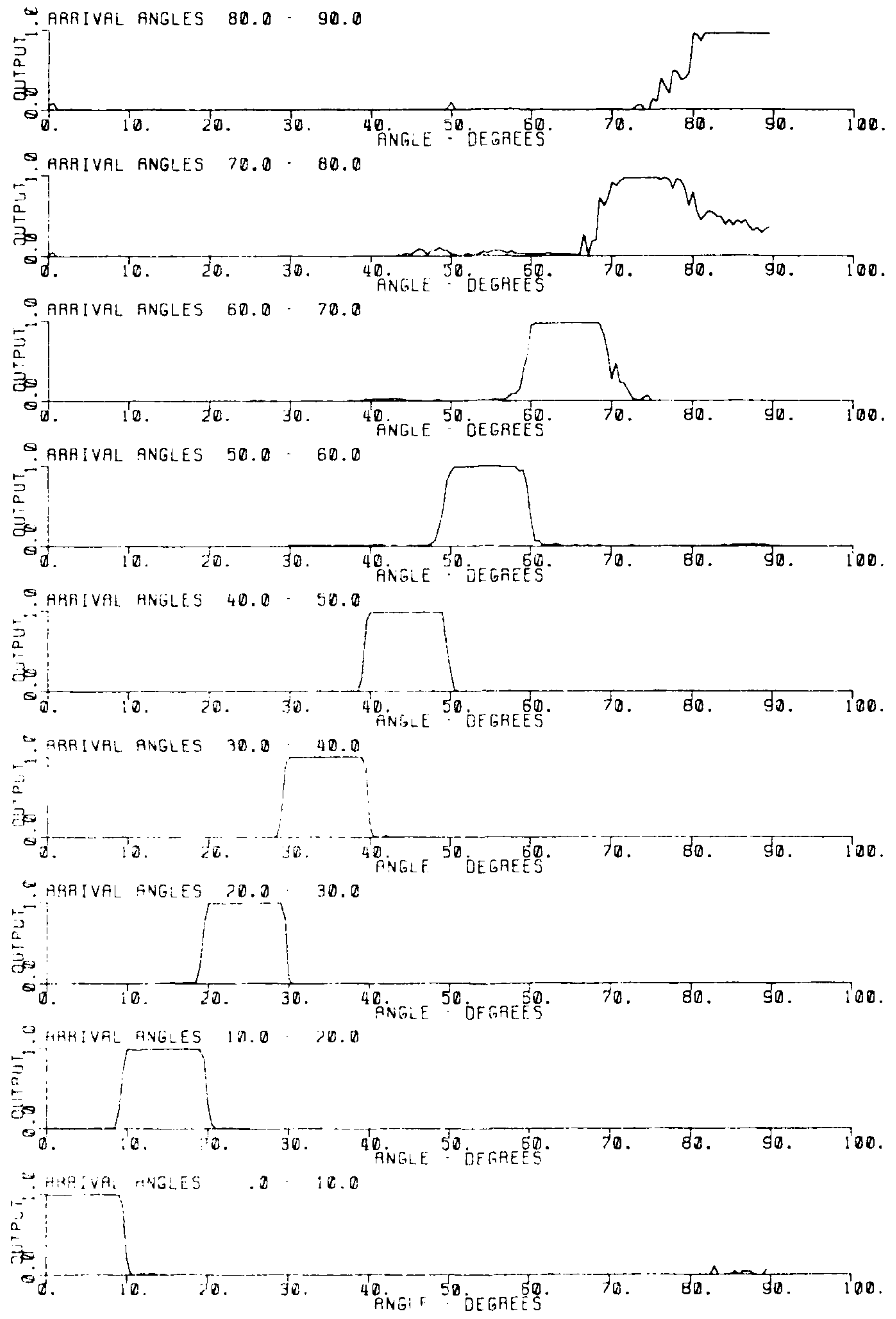


Figure 42

Exact Simulation; 10 Inputs; Amplitude Range .5-1.0

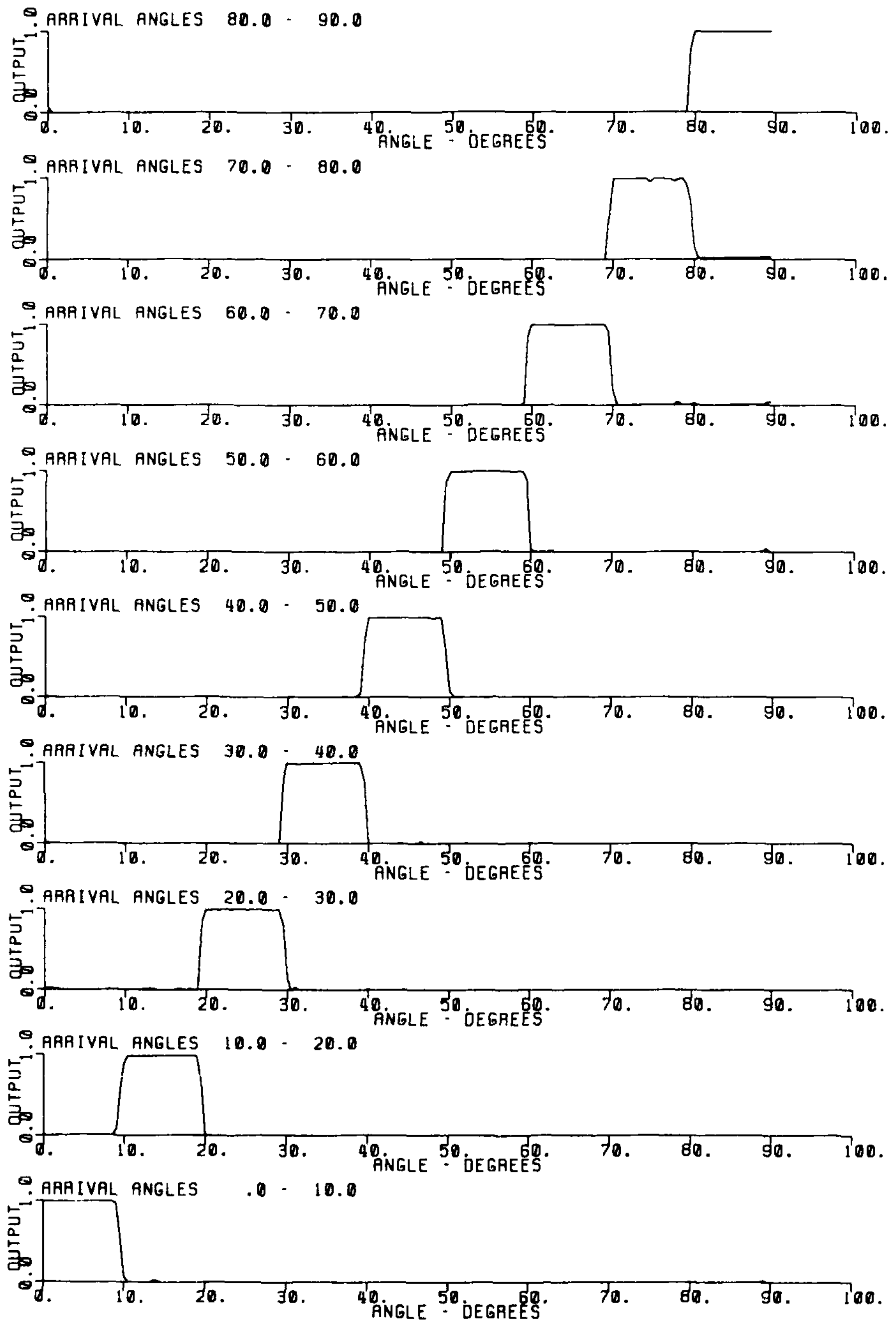


Figure 43

Exact Simulation; 20 Inputs; Amplitude Range .5-1.

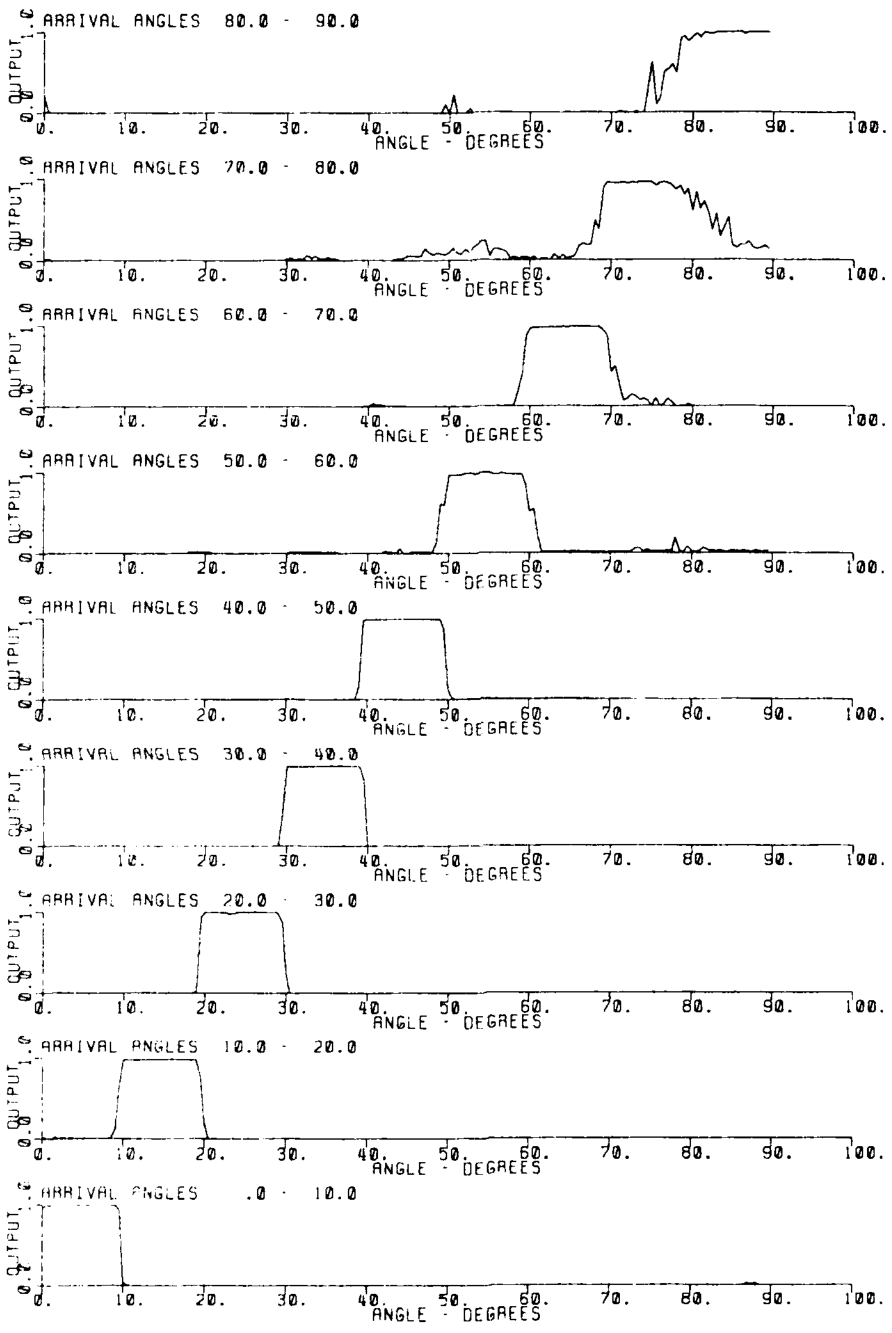


Figure 44

Exact Simulation; 10 Inputs; Amplitude Range .5 to 2.

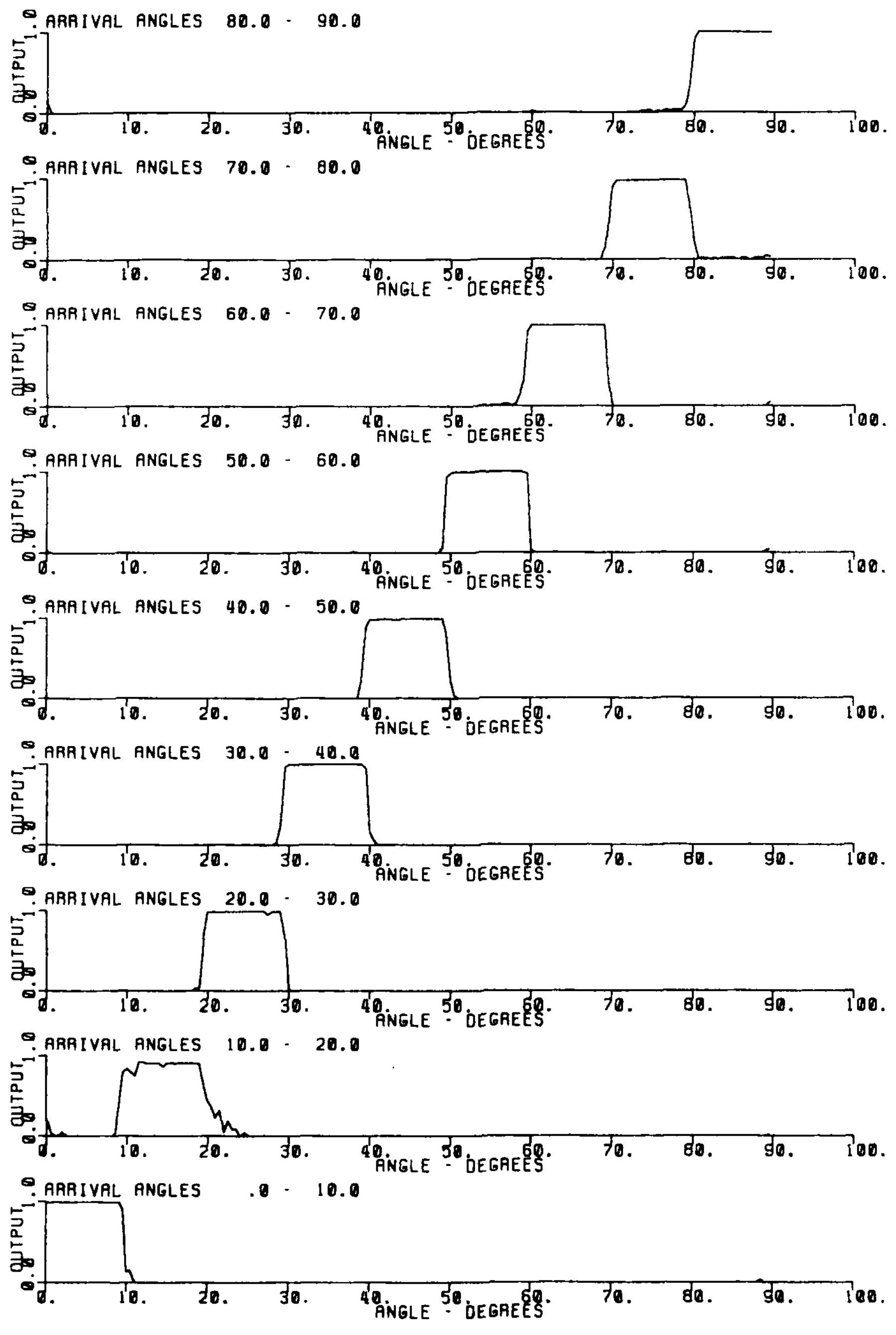


Figure 45

Exact Simulation; 20 Inputs; Amplitude Range .5-2.

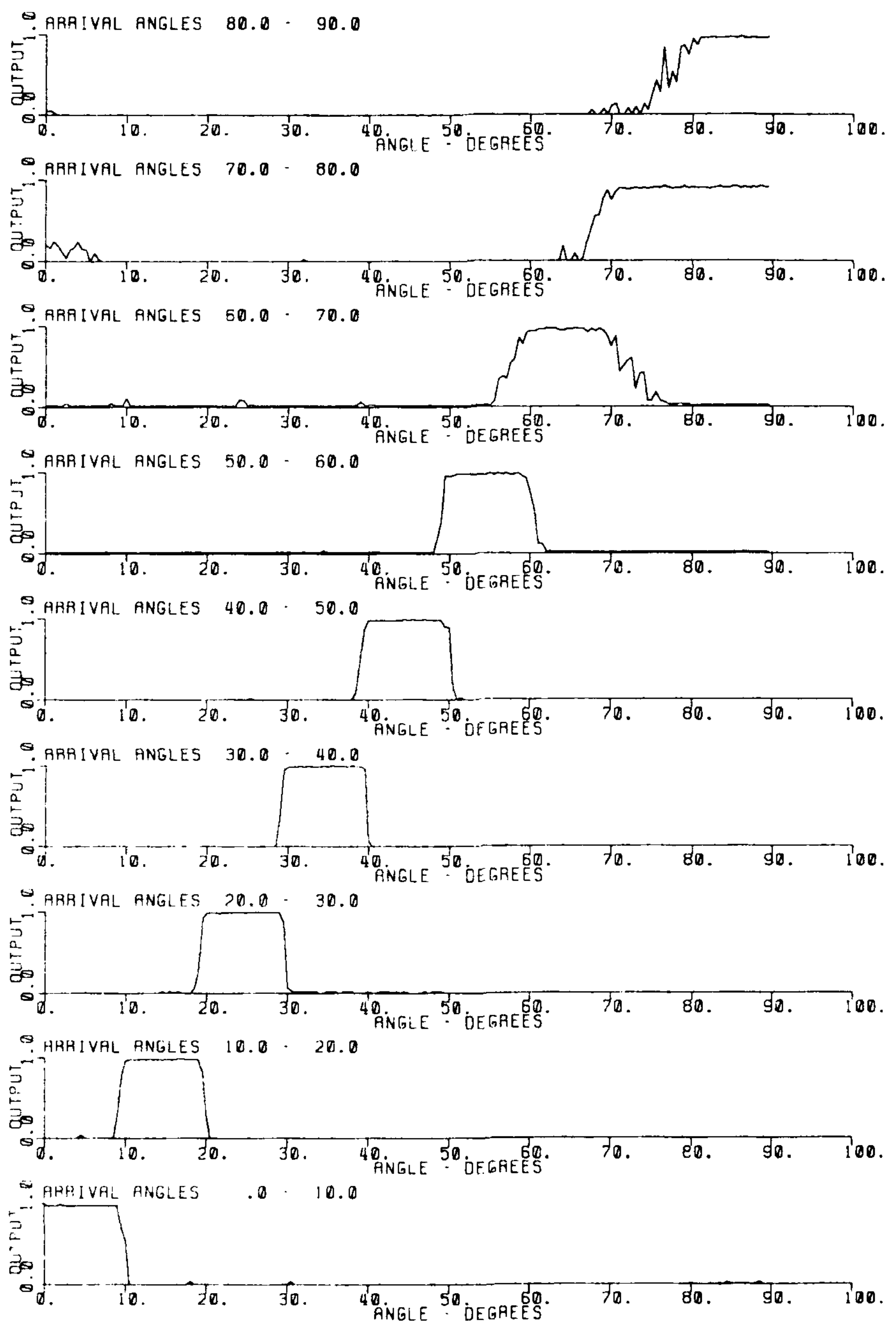


Figure 46

Elliptical Simulation; 10 Inputs; Amplitude Range of .5-1.

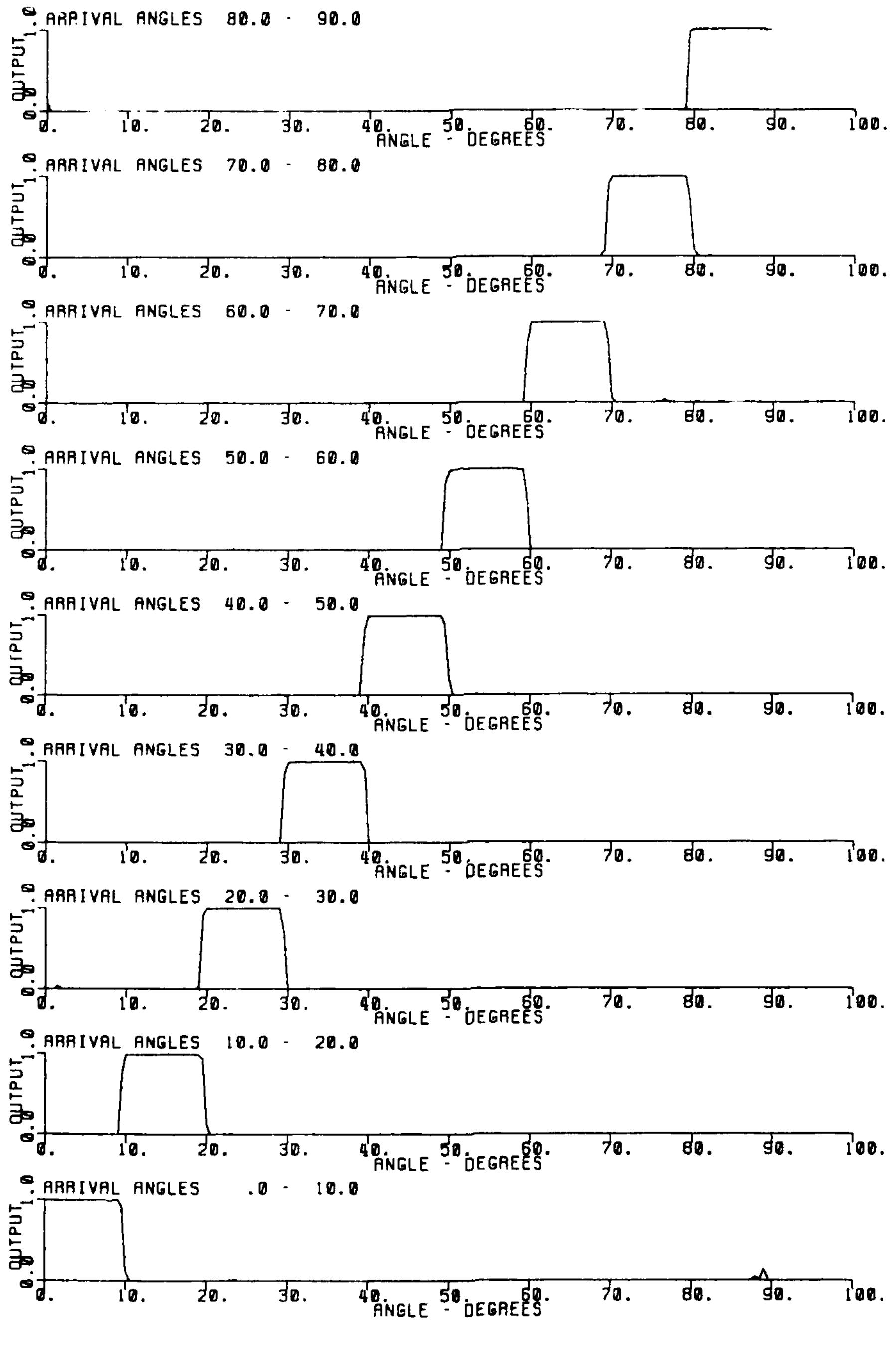


Figure 47

Elliptical Simulations; 20 Inputs; Amplitude .5-1.0

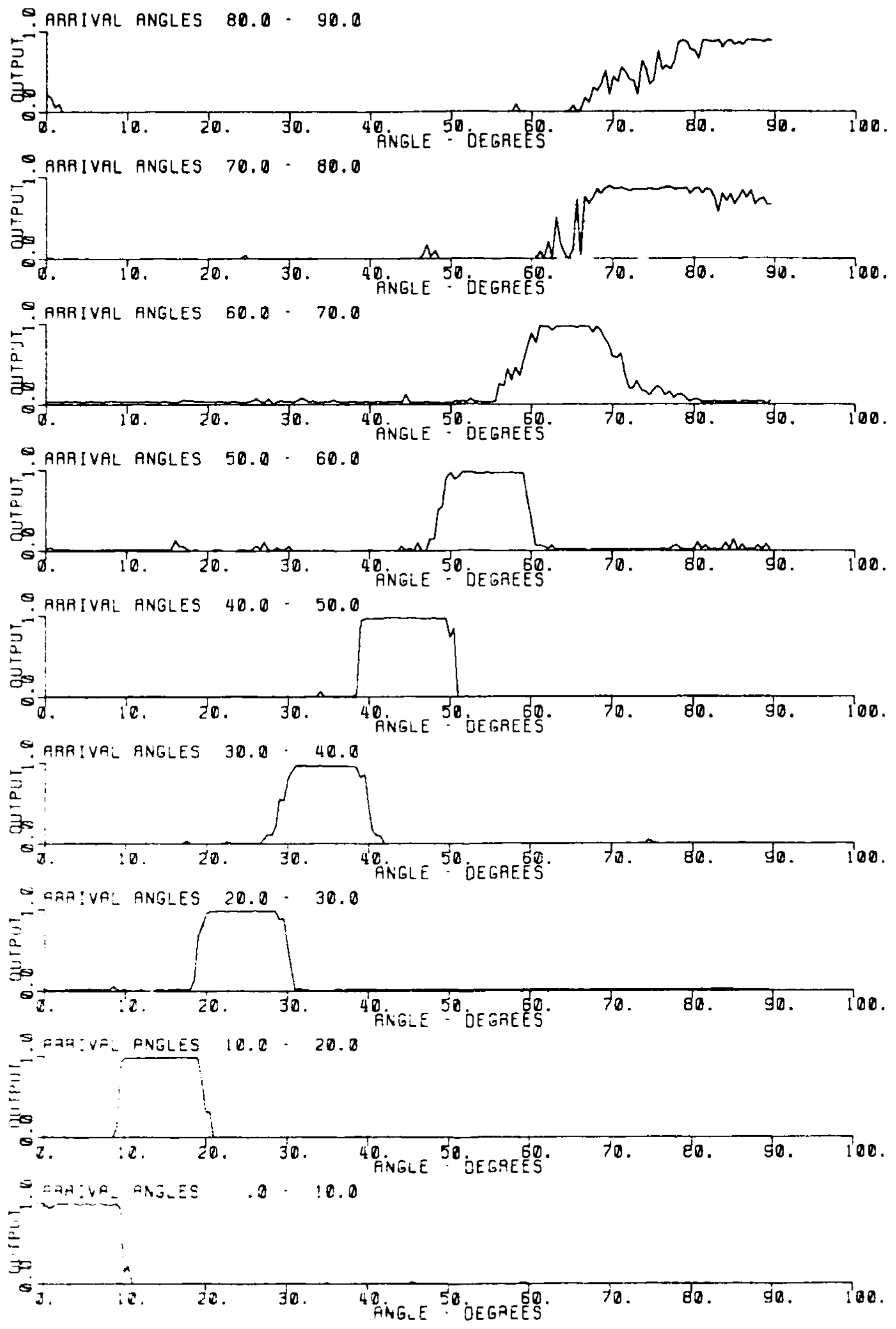


Figure 48

Elliptical Simulation; 10 Inputs; Amplitude Range of .5-2.

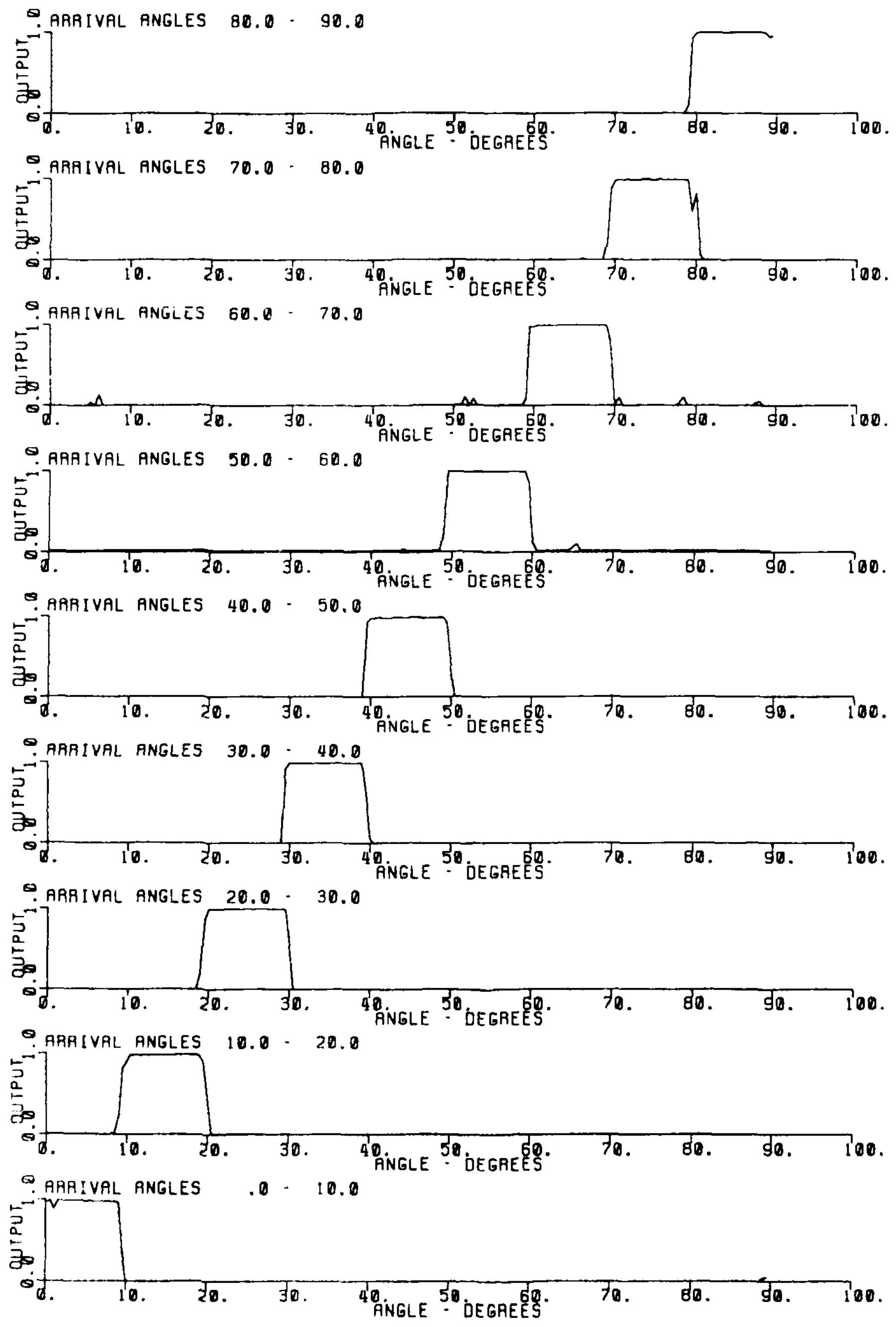


Figure 49

Elliptical Simulation; 20 Inputs; Amplitude Range .5-2.

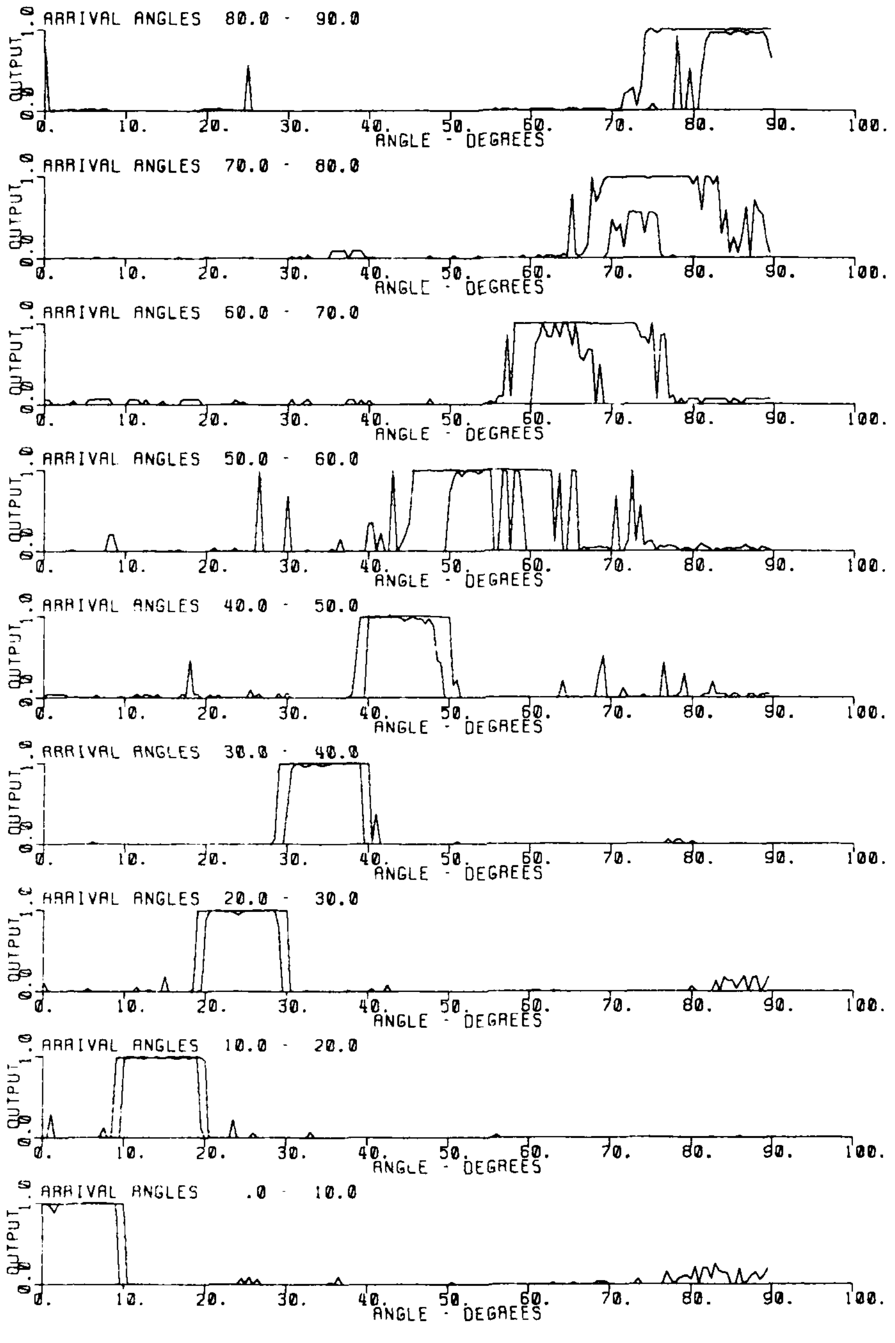


Figure 50

Simulation with Linear Inputs

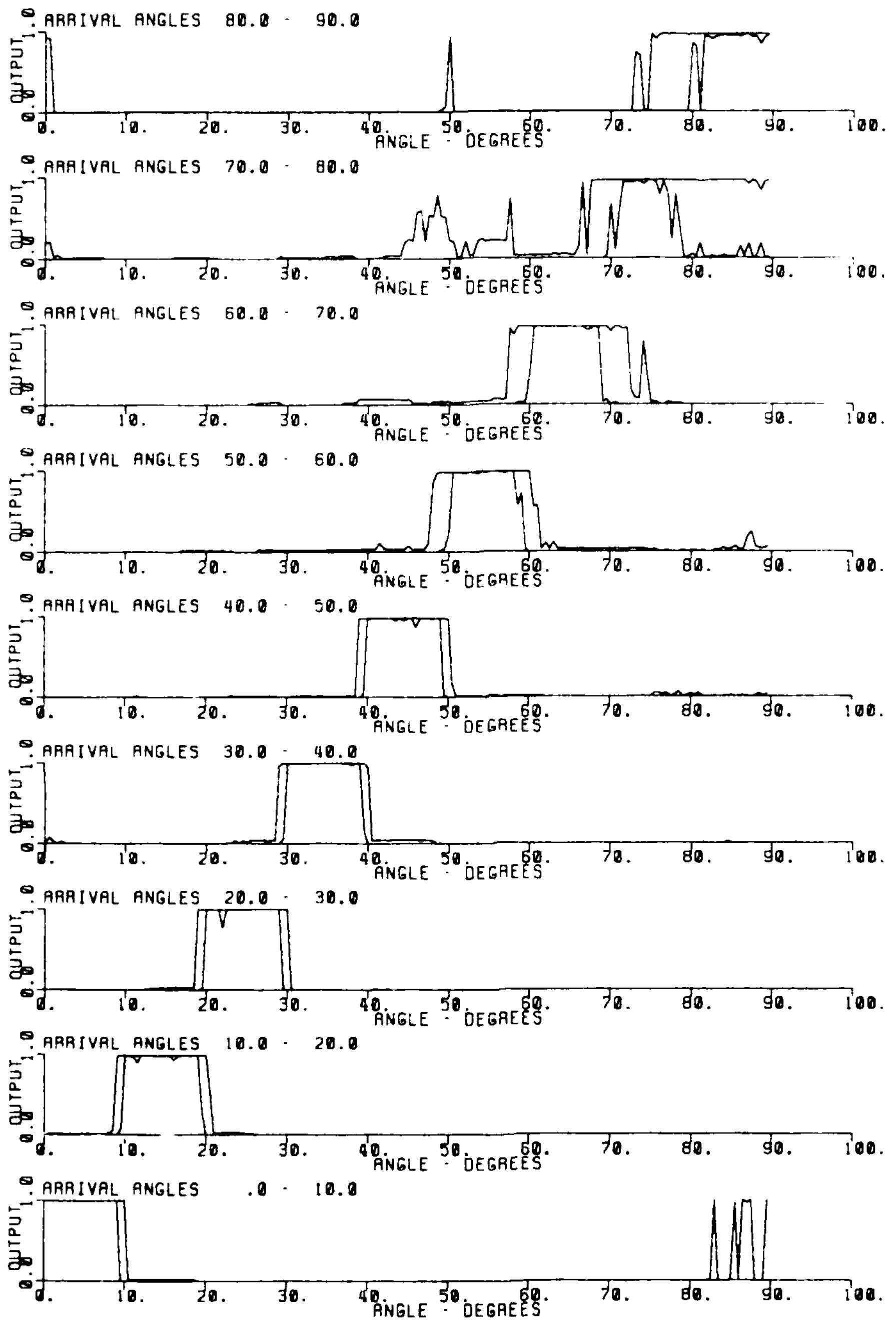


Figure 51

Simulation with Nonlinear Inputs

Chapter 5

Model of Wideband Beamformer

5.1 Design of Wideband Beamformer

When designing a wideband beamformer more information must be input into the network in order to determine direction correctly. This can be done in one of two ways. Temporal data can be saved and applied to another dimension of neurons in the network, or a two dimensional array of sensors can be used. The choice may depend on the price of sensors versus the price of analog circuits to store analog outputs. For this discussion the two dimensional beamformer array will be used. An example of a typical array is shown in Figure 52.

To determine direction from this array it can be viewed as two arrays, one in each direction. It was shown in Chapter 3 that a wideband beamformer is a function of both ω and θ. Since two variables are involved two linearly independent equations in the two unknowns are required.

5.2 Mathematical Analysis of Wideband Beamformer

An analysis of a wideband beamformer is similar to that of a narrowband beamformer. For the units in one dimension

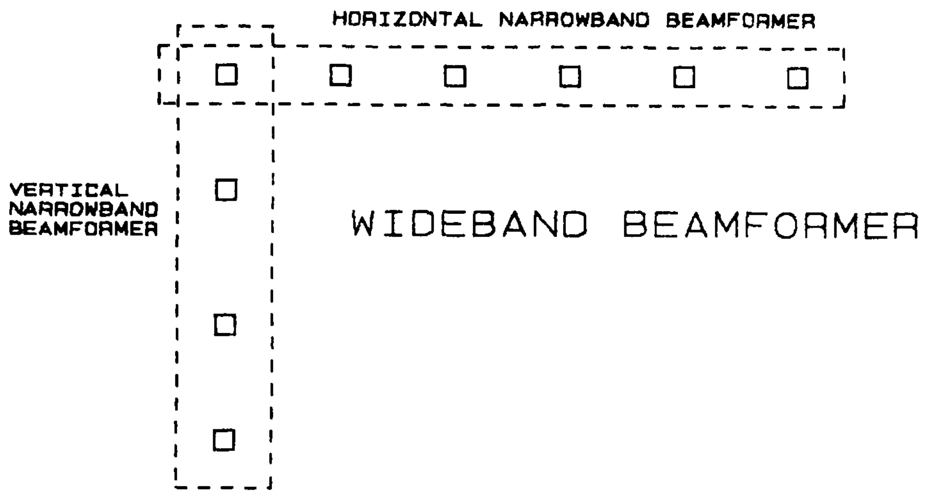

HORIZONTAL NARROWBAND BEAMFORMER
VERTICAL NARROWBAND BEAMFORMER
WIDEBAND BEAMFORMER

Figure 52

Two Dimensional Beamformer

the beamformer is the same as before.

$$V_i = A \cdot SIN(wt + i\alpha) \tag{5.1}$$

$$\alpha = \frac{2\pi d SIN(\theta)}{\lambda} \text{ and } \lambda = \frac{2C\pi}{w}$$

Where C is the propagation Velocity.

In the direction perpendicular to this line the formula will be

$$V_j = A \cdot SIN(wt + j\beta) \tag{5.2}$$

$$\beta = \frac{2\pi d COS(\theta)}{\lambda}.$$

As was shown in Equation 4.3 the equations for V_0 and V_1 can be combined to form an equation that is independent of time.

$$V_1^2 - 2V_0V_1 COS(gw) + V_0^2 - A^2 SIN^2(gw) = 0 \tag{5.3}$$

The second sensor's contribution to the beamformer can also be written as:

$$I_2^2 - 2I_0 I_2 \mathrm{COS}(2gu) + I_0^2 - A^2 \mathrm{SIN}^2(2gu) = 0. \qquad (5.3)$$

where $g = \frac{d}{c}\mathrm{SIN}(\theta)$.

The two Equations 5.1 and 5.2 are not linearly independent. Therefore the variables g or θ cannot be evaluated independently of ω. However if the second beamformer line is in another direction such that $g = \frac{d}{c}\mathrm{COS}(\theta)$, the twc equations can be combined to yield the following formula that is independent of ω (frequency) and t (time).

$$\mathrm{TAN}(\theta) = \frac{\mathrm{COS}^{-1}\left(\frac{-I_0 I_1}{A^2} \pm \sqrt{\left(\frac{I_0^2 I_1^2 - A^4 - A^2 I_1^2 - A^2 I_0^2}{A^4}\right)}\right)}{\mathrm{COS}^{-1}\left(\frac{-I_0 I_2}{A^2} \pm \sqrt{\left(\frac{I_0^2 I_2^2 - A^4 - A^2 I_2^2 - A^2 I_0^2}{A^4}\right)}\right)} \qquad (5.6)$$

This formula demonstrates that when the 3 sensors are not in a straight line the resulting signal can be expressed in a manner that is independent of both time, t, and frequency, ω. However it is not in a form that can be represented in one neuron as was the case with the narrowband beamformer. It is known that any function can be represented in three levels of neurons if enough hidden units are used. Therefore backpropagation can be used to train the network to determine the weights of the network.

5.3 Wideband Beamformers Using Backpropagation

Wideband Beamformers can be trained in several ways. Just as with the narrowband beamformer these networks can be trained by using the Delta Rule where the output of each level is known. However unlike the narrow band beamformer the appropriate values for the hidden units are not easy to approximate. Mathematical calculations of the weight would also be very difficult. This makes backpropagation the best choice for training the network.

5.4 Results of Beamformers Trained with Backpropagation

Several sets of plots are presented in which Backpropagation was used to train the network. The backpropagation required over 60,000 iterations of training.

These plots are shown in Figures 53 through 61. In these simulations three levels of perceptrons were used to form the network. The simulations are made in sets of three to provide an example of linear, exact and elliptical networks. These sets of simulations are performed for networks with three hidden units in each level (3x3) , 6x6 hidden units, and 10x10 hidden units. These simulations indicate that linear networks actually outperform the elliptical and exact networks. The nonlinear inputs helped when simple narrowband cases were used, but as the complexity of the input signal increased the linear model proved to be the best. The advantage of using more hidden units is also demonstrated. A comparison between these three types of networks and the number of hidden units is shown in Table 5.

Network Hidden Units	Linear Network % Correct	Exact Network % Correct	Elliptical Network % Correct
3x3	91.87	81.08	80.96
6x6	94.29	93.40	87.12
10x10	96.62	96.02	96.04

Table 5

Comparison of Wideband Beamformer Networks

All of the previous plots were made using a single frequency component that was allowed to vary over a wideband as the system was trained. To better demonstrate the networks wideband abilities simulations were made in which two frequency components of random amplitude were allowed to vary as the network was trained. After training, simulations were made in which 1, 2, and 7 frequency components were used to determine whether the network was sensitive to the spectra of the waveform. These simulations are shown in Figures 62 through 64. These plots demonstrate that the network is sensitive to phase and not the spectra of the waveform.

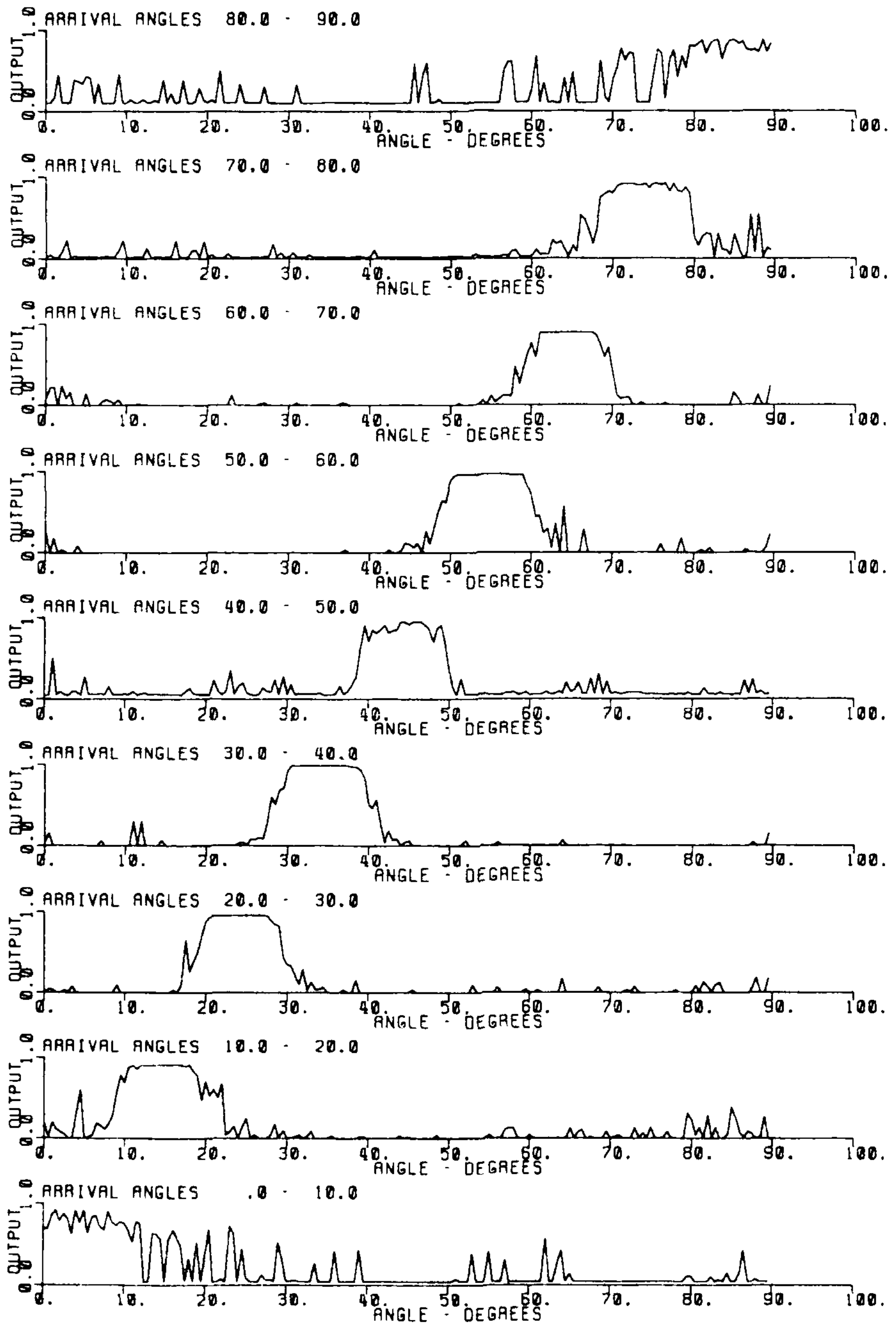


Figure 53

Linear Wideband ANN, 3x3 Hidden Units

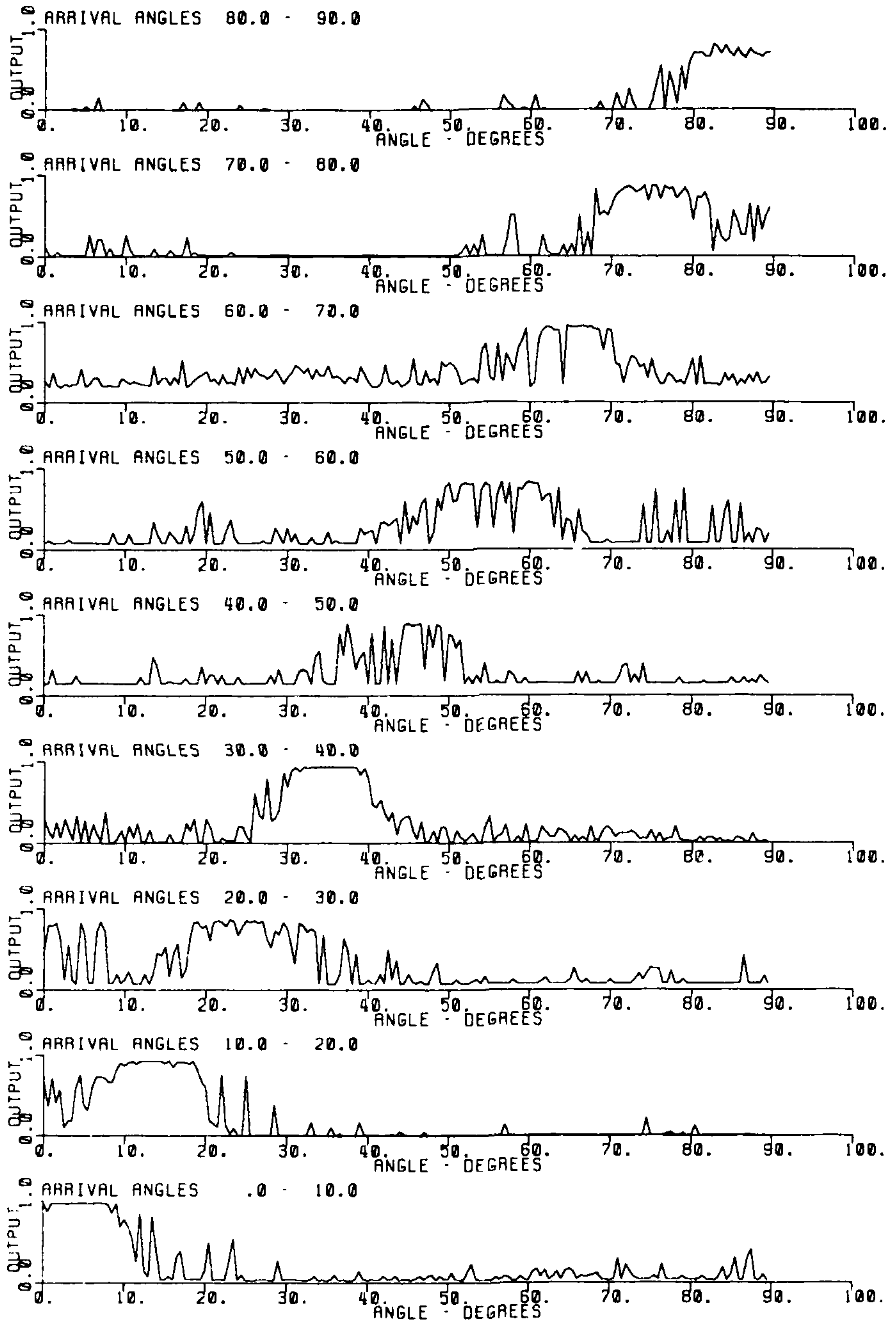


Figure 54

Exact Wideband ANN, 3x3 Hidden Units

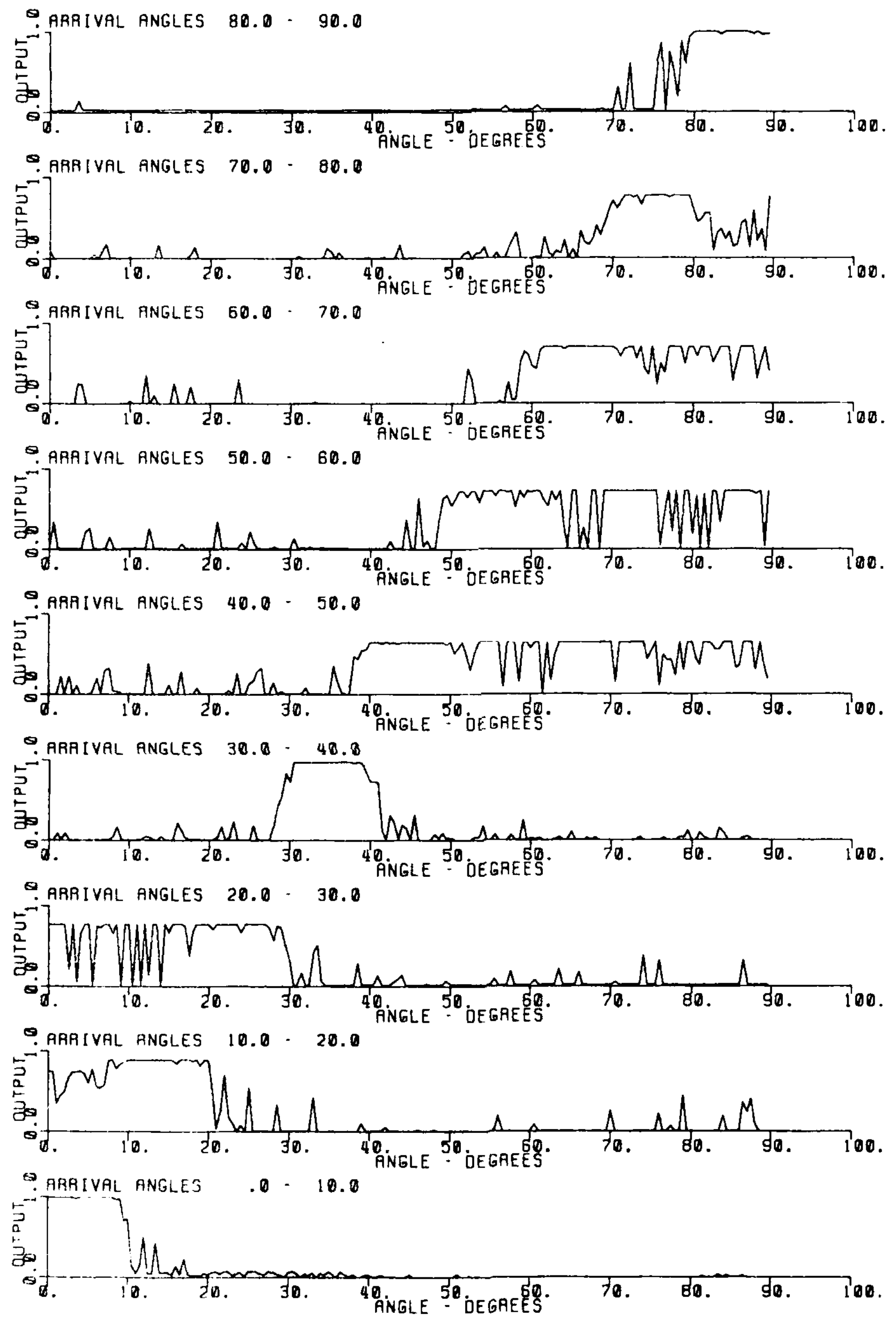


Figure 55

Elliptical Wideband ANN, 3x3 Hidden Units

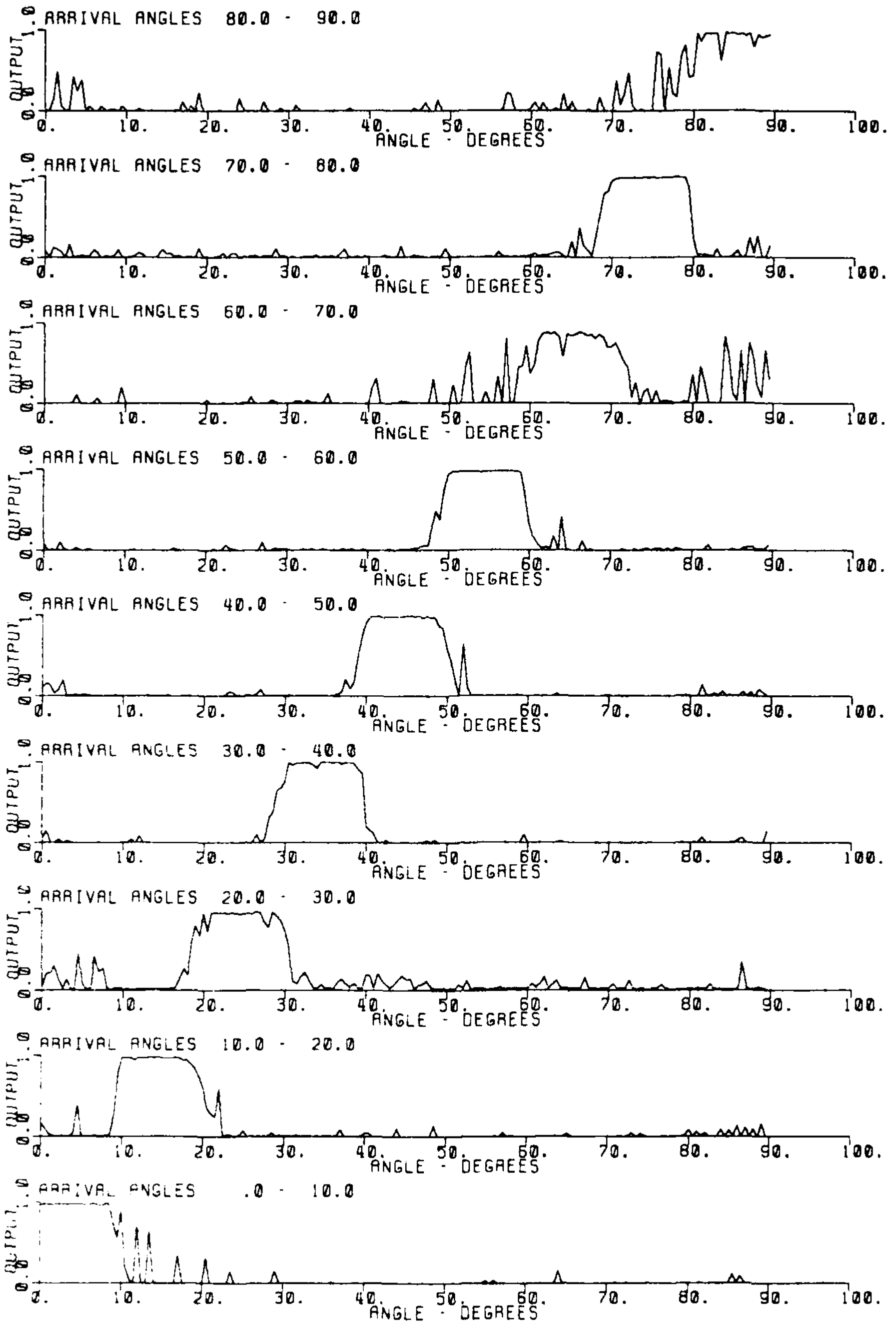


Figure 56

Linear Wideband ANN, 6x6 Hidden Units

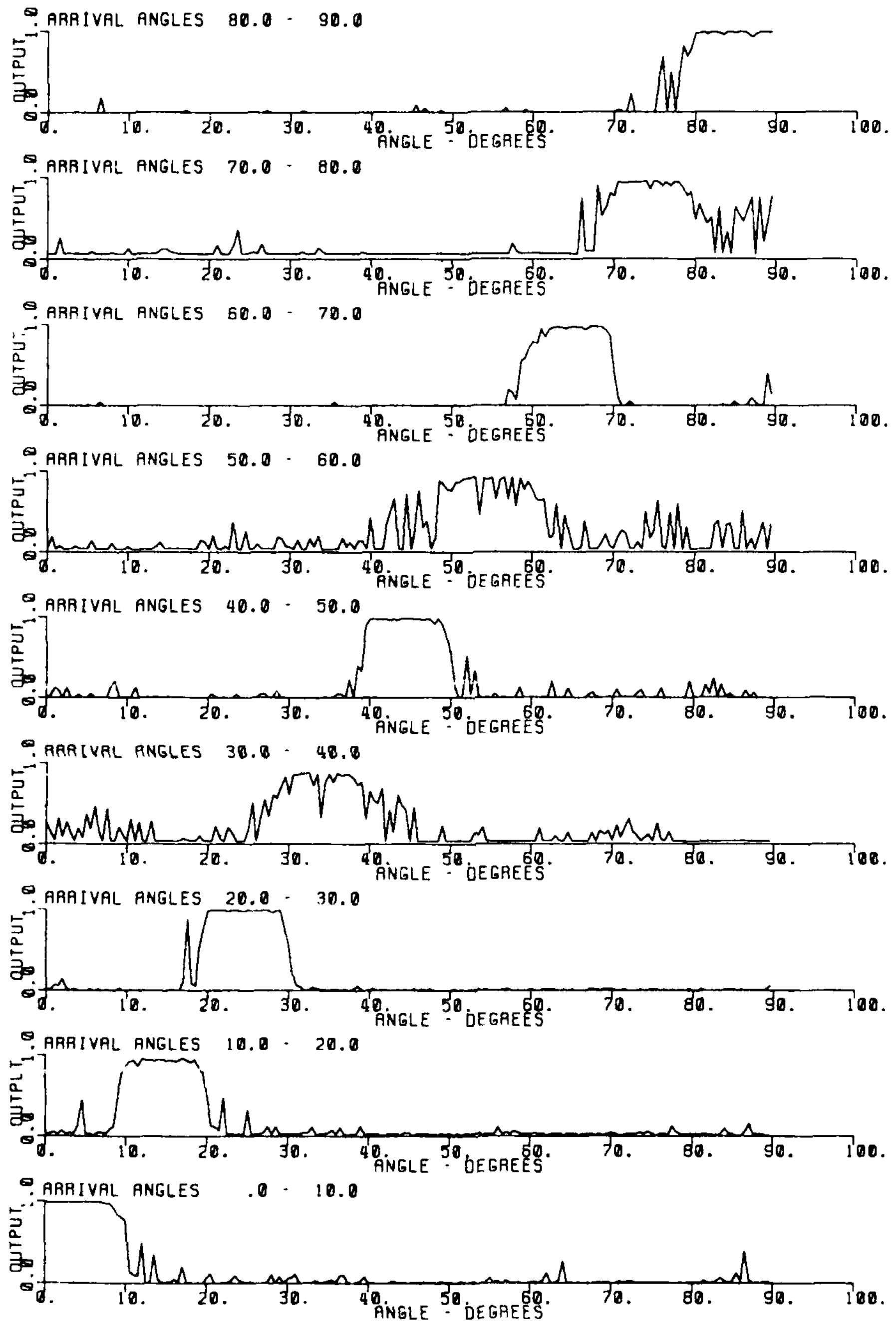


Figure 57

Exact Wideband ANN, 6x6 Hidden Units

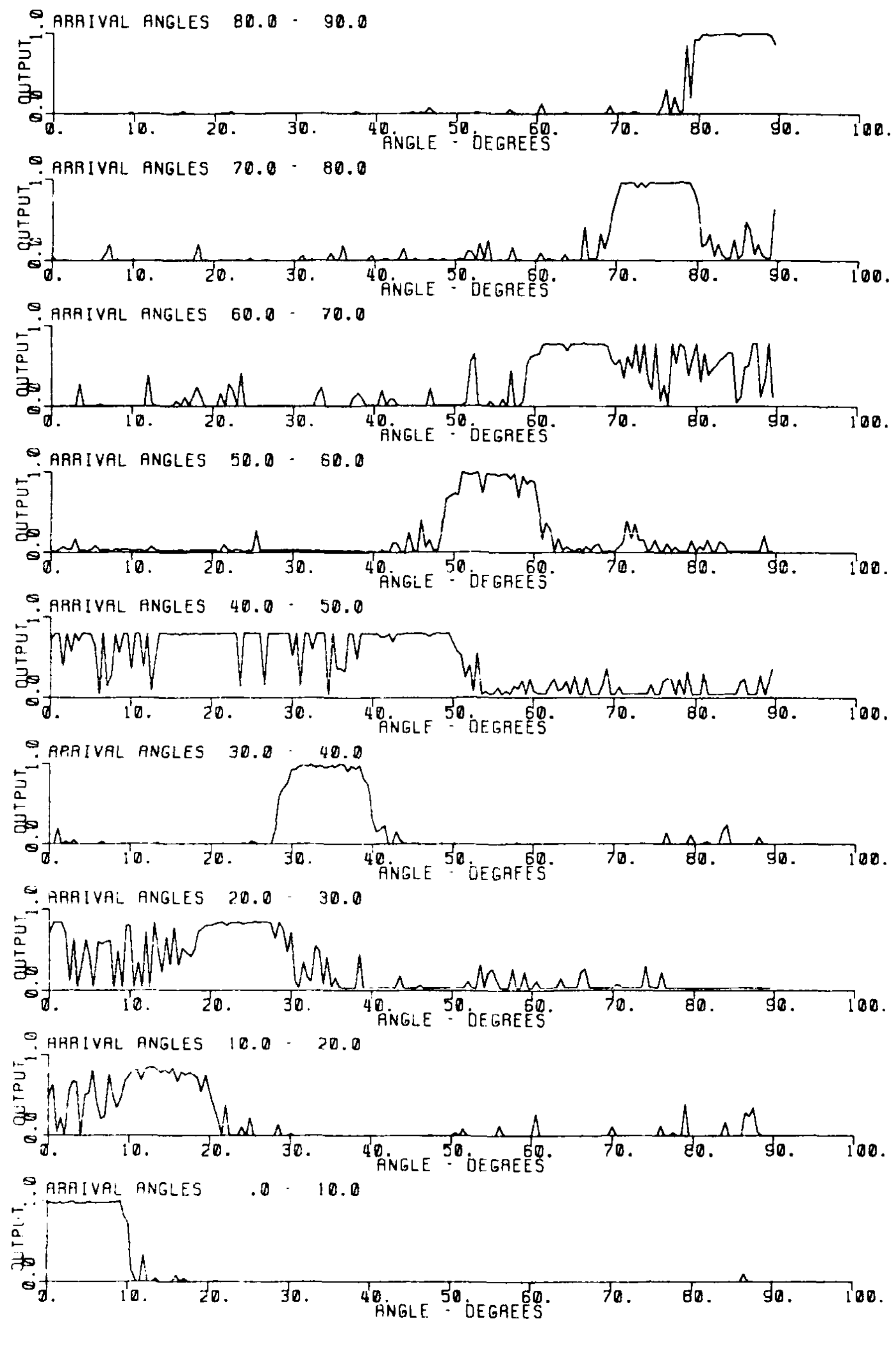


Figure 58

Elliptical Wideband ANN, 6x6 Hidden Units

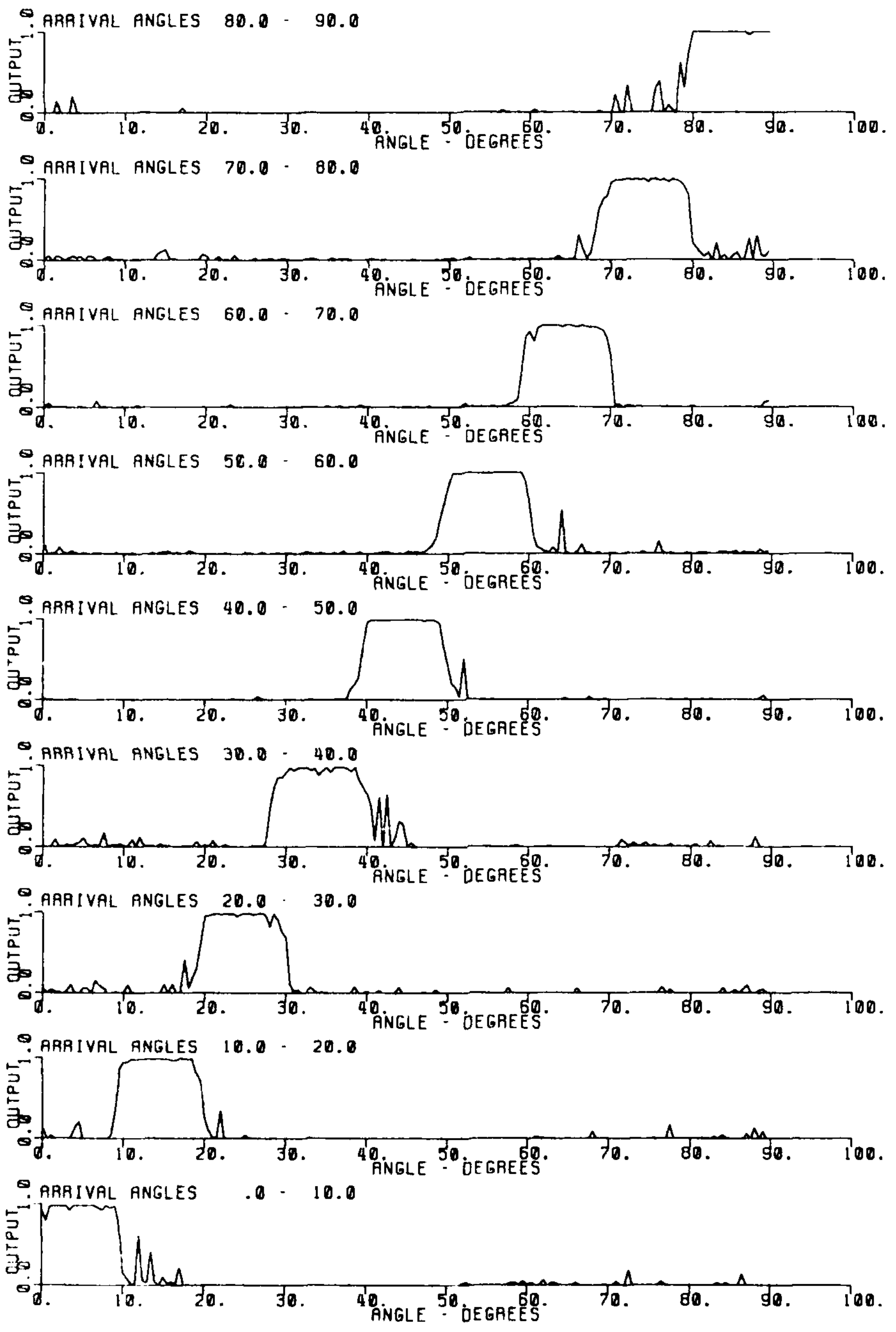


Figure 59

Linear Wideband ANN, 10x10 Hidden Units

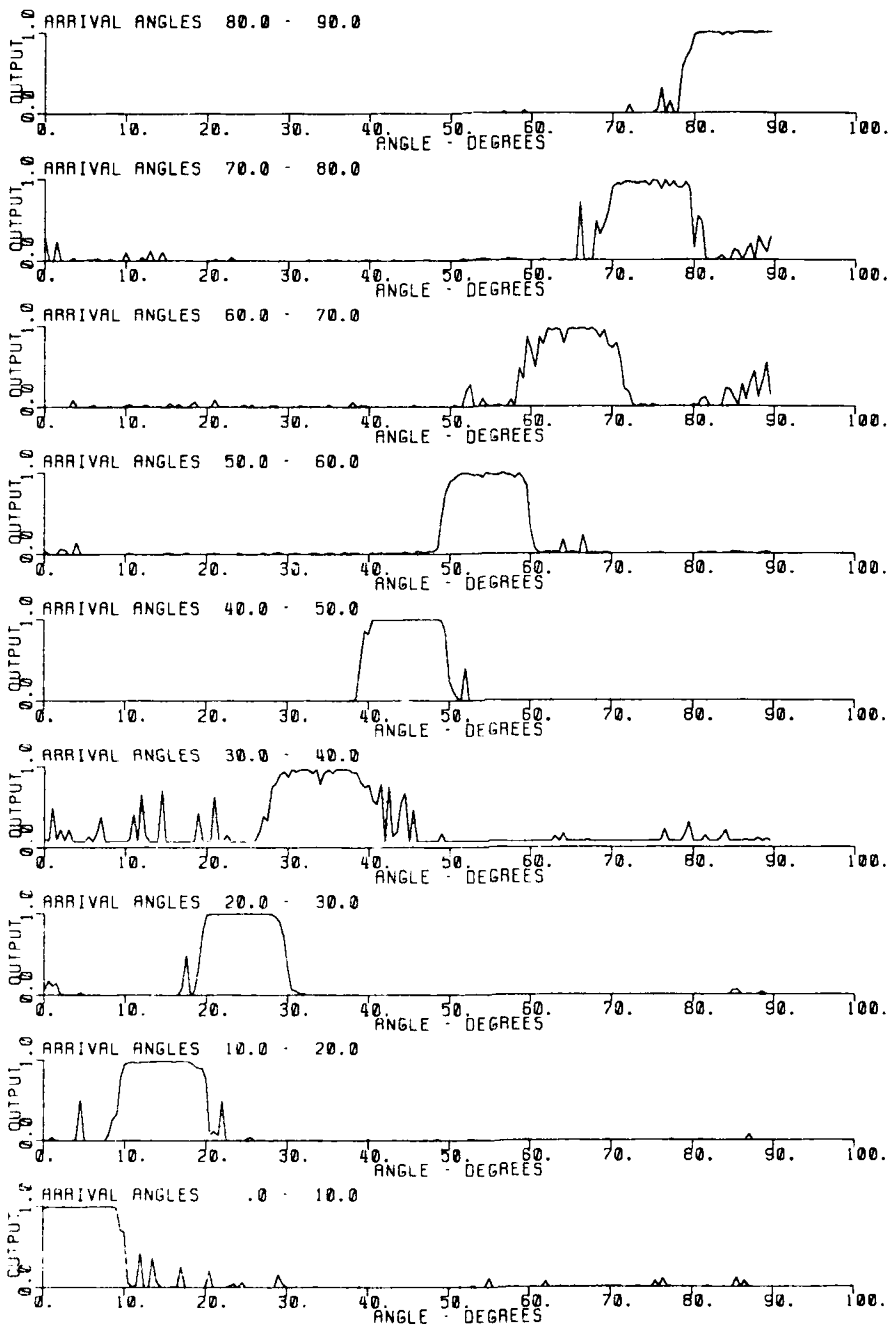


Figure 60

Exact Wideband ANN, 10x10 Hidden Units

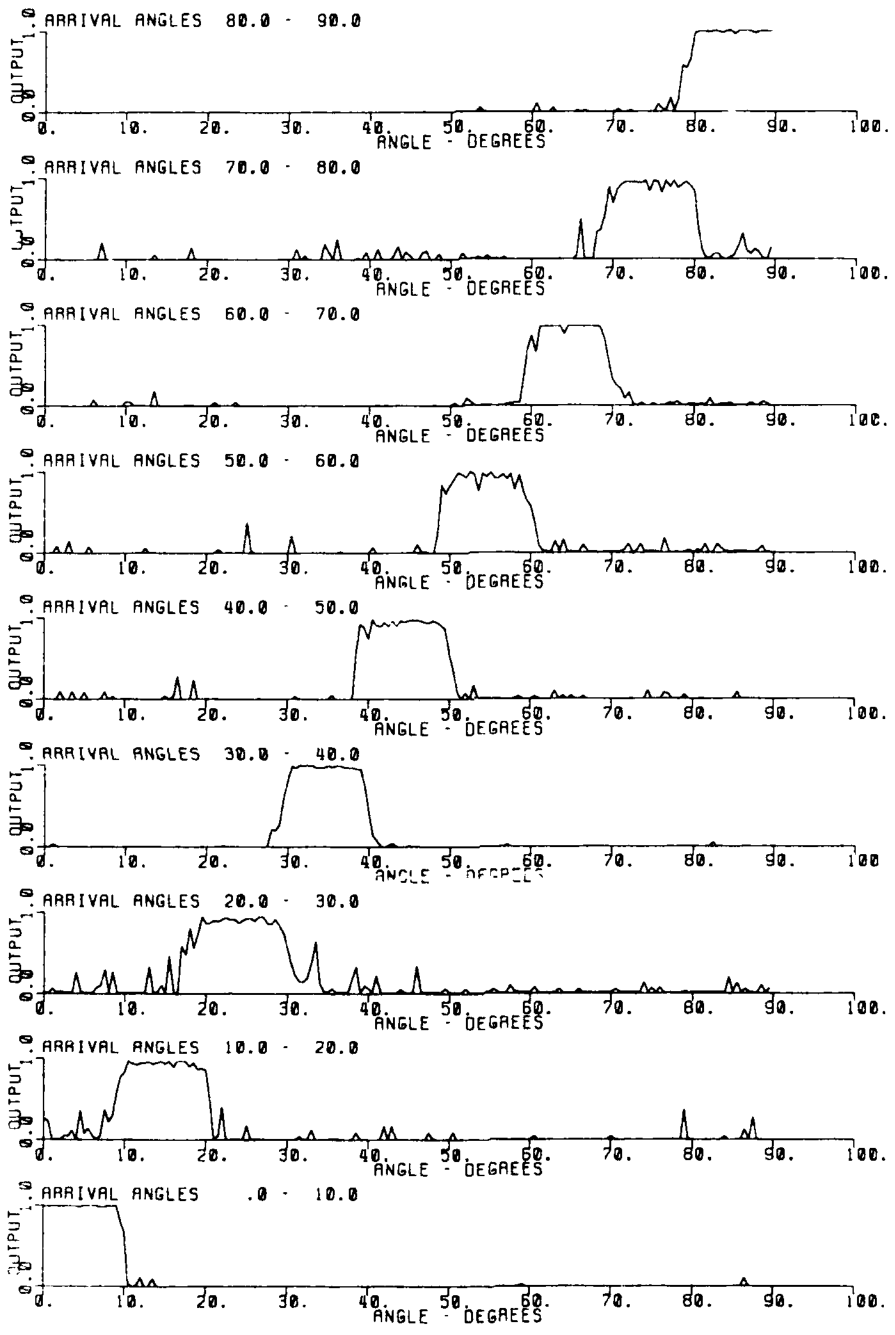


Figure 61

Elliptical Wideband ANN, 10x10 Hidden Units

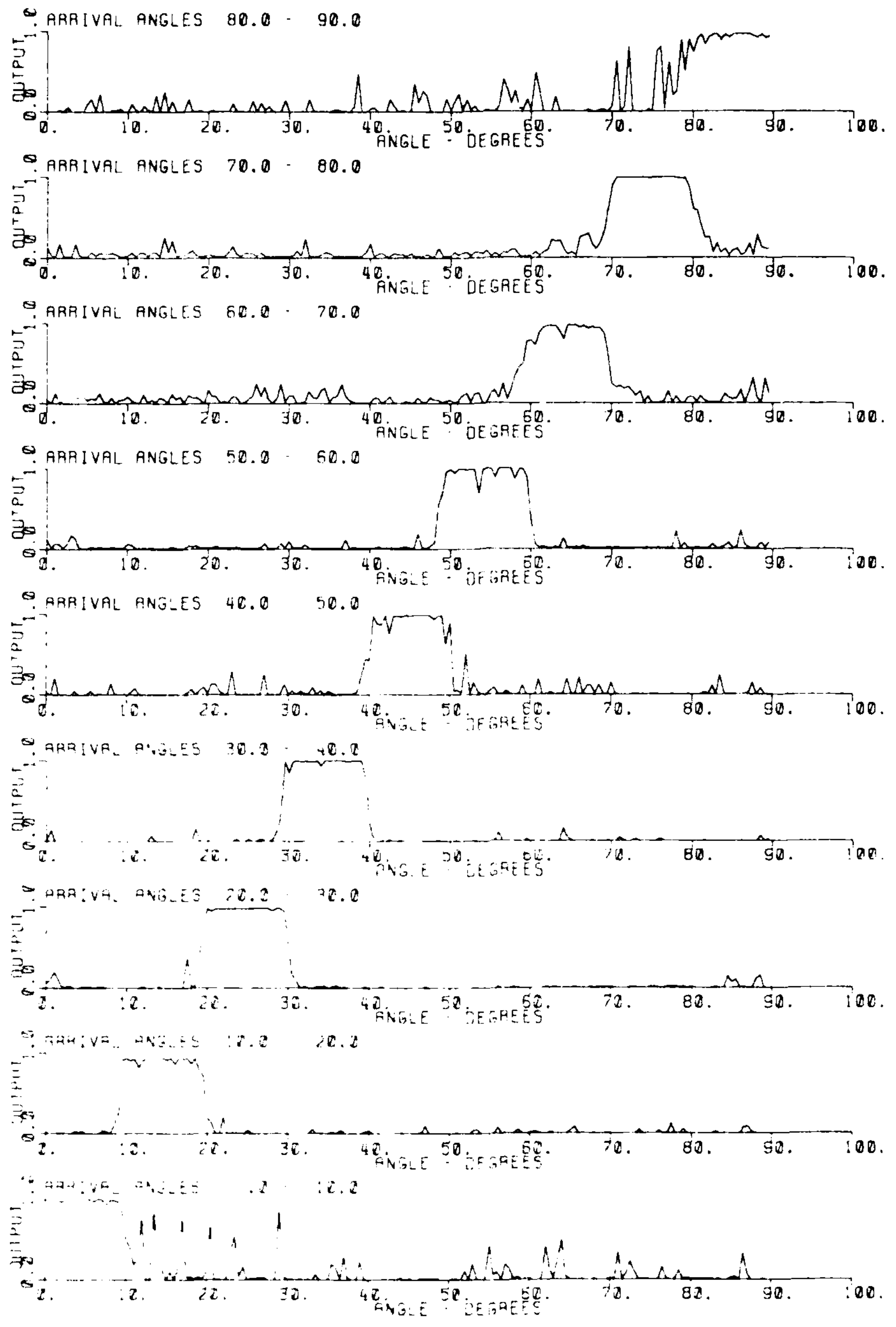


Figure 62

Linear Wideband ANN, 1 Frequency Component

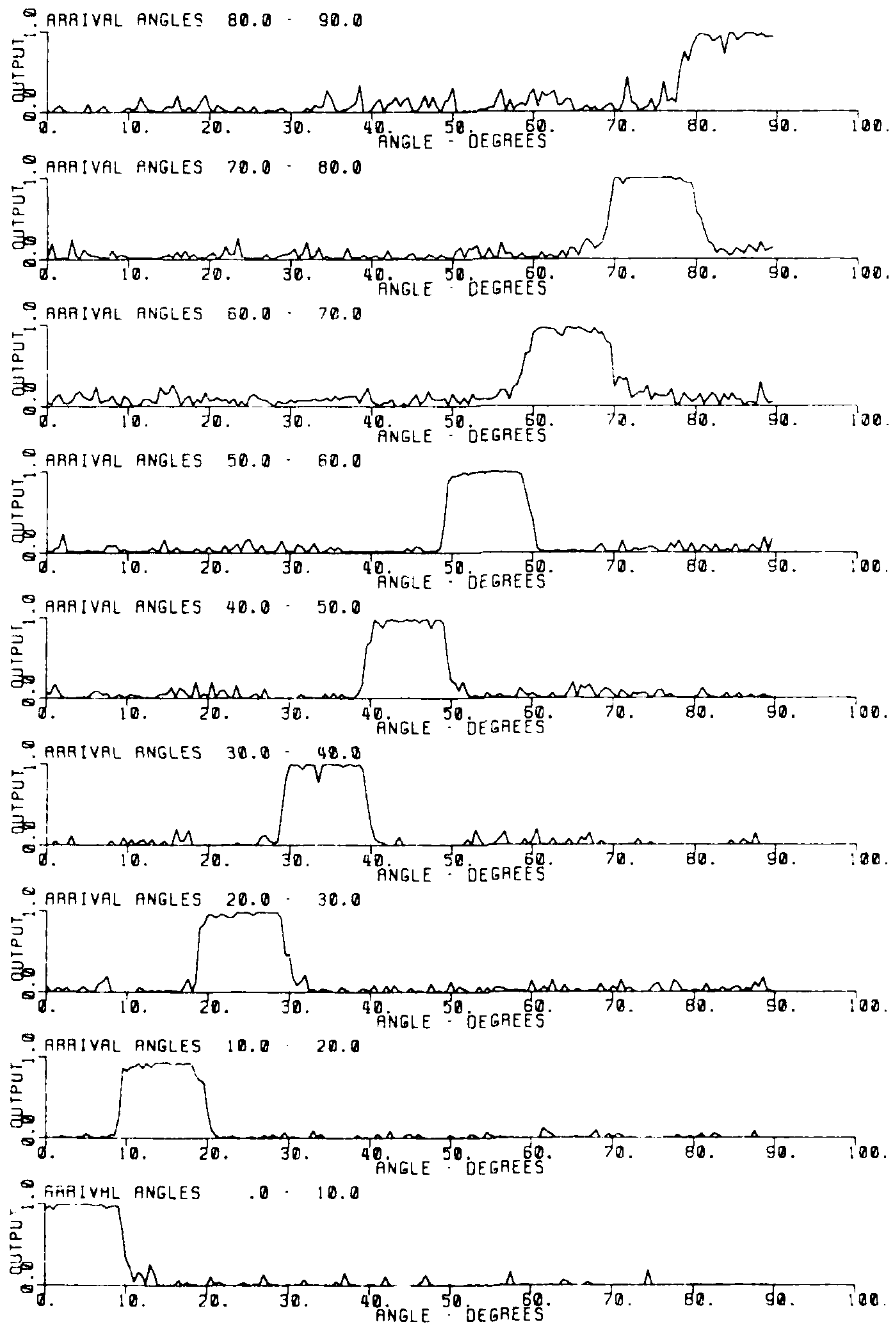


Figure 63

Linear Wideband ANN, 2 Frequency Components

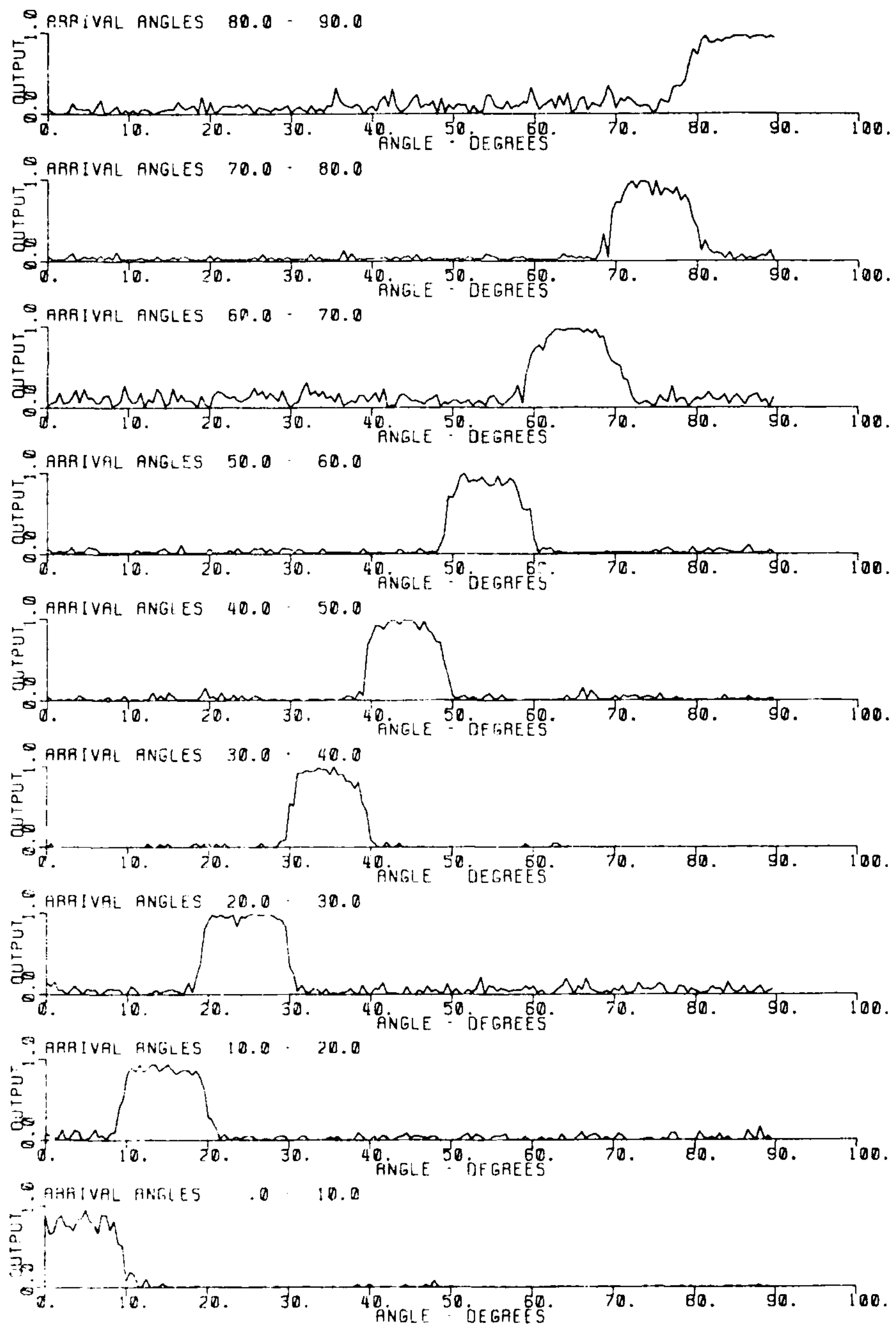


Figure 64

Linear Wideband ANN, 7 Frequency Components

Chapter 6

Empirical Demonstration and Comparison of FFT and ANN Beamformers

6.1 Seismic Test Description

To demonstrate how the direction of arrival of a signal can be determined from a trained ANN, the following experiment was performed. A seismic array was arranged in a three dimensional pattern with twelve geophones. A movable signal source was used to transmit a wideband wave from three different directions. The received signals were corrupted by the presence of a crane at a fixed position in the vicinity of the geophones. In addition to this mechanical noise, 60 HZ electrical noise also corrupted the data. Several tests in which the signal source was moved to different ranges in the three directions were recorded and digitized. A diagram of the test is shown in Figure 65.

The task for the neural network to perform is one of determining the direction of the signal while rejecting the mechanical and electrical noise in the system. One of the advantages which neural networks have over conventional signal

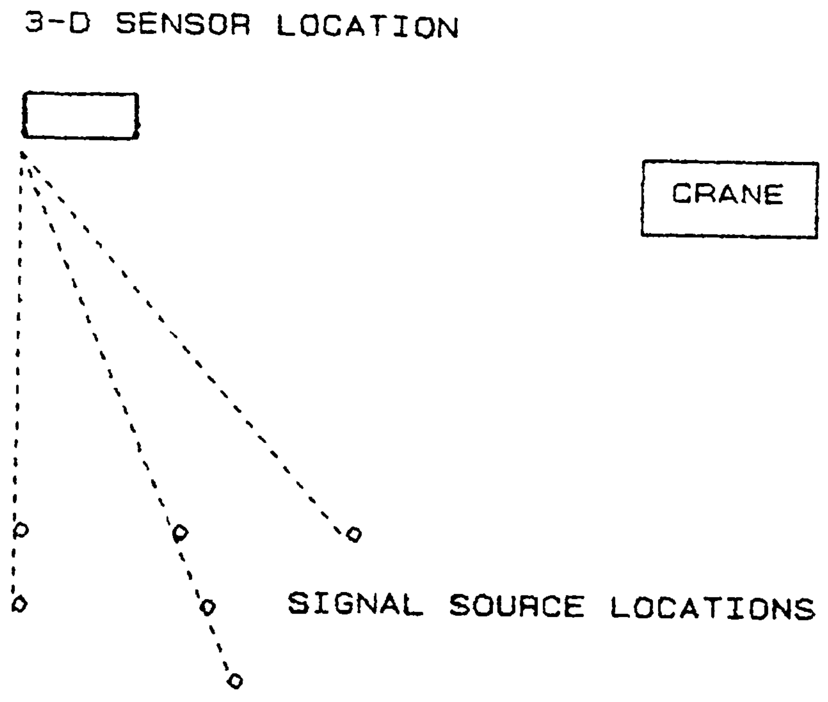


Figure 65

Seismic Test Diagram

processing is the small amount of information required to train the network. For example the exact directions are not known but are labeled A, B, C for convenience. The array's configuration is known to be three dimensional but the order and exact location of the gages need not be known. The calibration of the gages in inches per second is known to be sufficient to acquire the signal of interest but the output is passed through an automatic gain control routine to provide the network with an input with a narrow amplitude range. The exact output level of the AGC is not known or required. The exact frequency spectra of the noise source and signal source are not known. However the data was digitized at a sampling rate greater than twice the bandwidth of the velocity gages used to acquire the data. This prevents aliasing of the time sampled data. Since the frequency spectra is not known, it was assumed to be a wideband signal. In order to prevent aliasing in the spatial dimension the spacing of some of the geophones were placed closer than one half of the minimum wavelength to be received. The minimum wavelength is equal to the propagation velocity divided by the maximum frequency that the gages can receive. With only a brief (20 seconds) time history of the output of the geophones the network can be trained to recognize the direction of the desired signal and ignore all other interference. A time history plot of the data used to train the network is shown in Appendix II. These plots show an example of one second of data at all three directions and two different ranges.

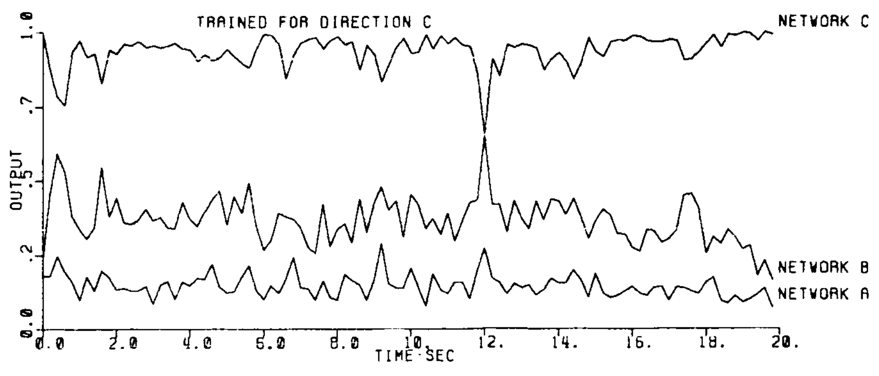
6.2 Results of Seismic Test

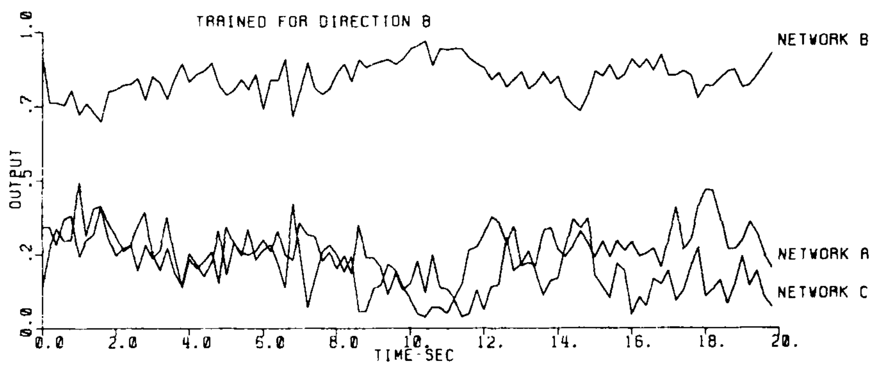
Once the network was trained samples of both the time signals used to train the network and the other recorded signals were processed through the network. The results are shown in Figures 66-71. Figure 66 is a simulation in which the input data was the same data that was used to train the network. This data was 20 seconds in duration and represented the data closest to the geophones. In Figure 67 another sample of data recorded at the same range was used for the input to the simulation. In Figure 68 through 71 samples of data from other ranges were used for the simulations. Each of the plots presents the three network outputs as a function of time. Ideally an output of one represents the presence of a signal from the direction associated with that output, and a zero represents the absence of a signal. Since only one signal was present in each of the tests, one output should predominate the other two.

These plots indicate that the response is much better for the data that trained the network than for other data sets. The expected reason for this is that the data used to train the network was not very representative of the data in the other data sets. The spectral characteristics of the data can easily change with distance or with the intermittent operation of the crane. It was demonstrated in Chapter 5 that when a wideband signal is used to train the network it will be able to recognize signals with different spectra. It is however imperative that training signals be wideband signals that encompass all the frequencies to be received by the ANN.

The output of the ANN was analyzed as if the output level was connected in a winner-take-all configuration. In such a configuration only the output with the highest amplitude is asserted. The results can then be compared with the known direction to determine the percent error in the simulation. Thirteen segments of data were analyzed. The error for each of the thirteen segments is shown in Table 6.

It should be noted that the results ranged from very good (0 % error) to rather bad (31 % error). The 0 % error occurred when the training set was used as the input to the network. The high errors however occurred at both ranges and at all three angles. This indicated that range was not as an important factor as were other temporal phenomena. This can be seen rather clearly from Figure 69. It can be seen that during the first twelve seconds the network functioned very well, but something happened during the last eight seconds that alters the networks result. This could easily have been a modification





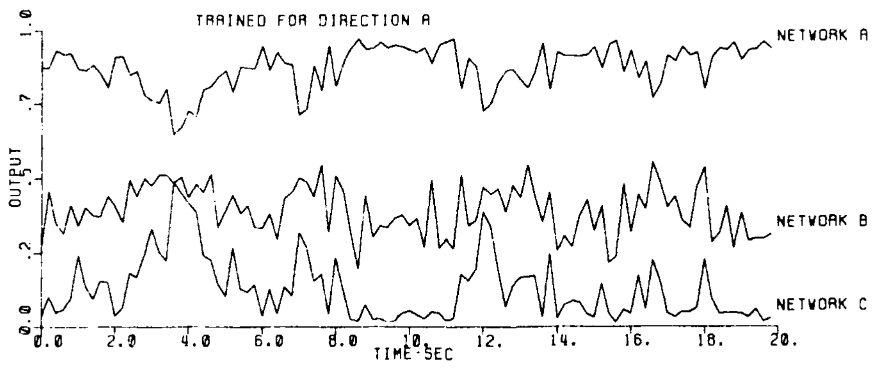


Figure 66

First Simulation with Training Data as Input

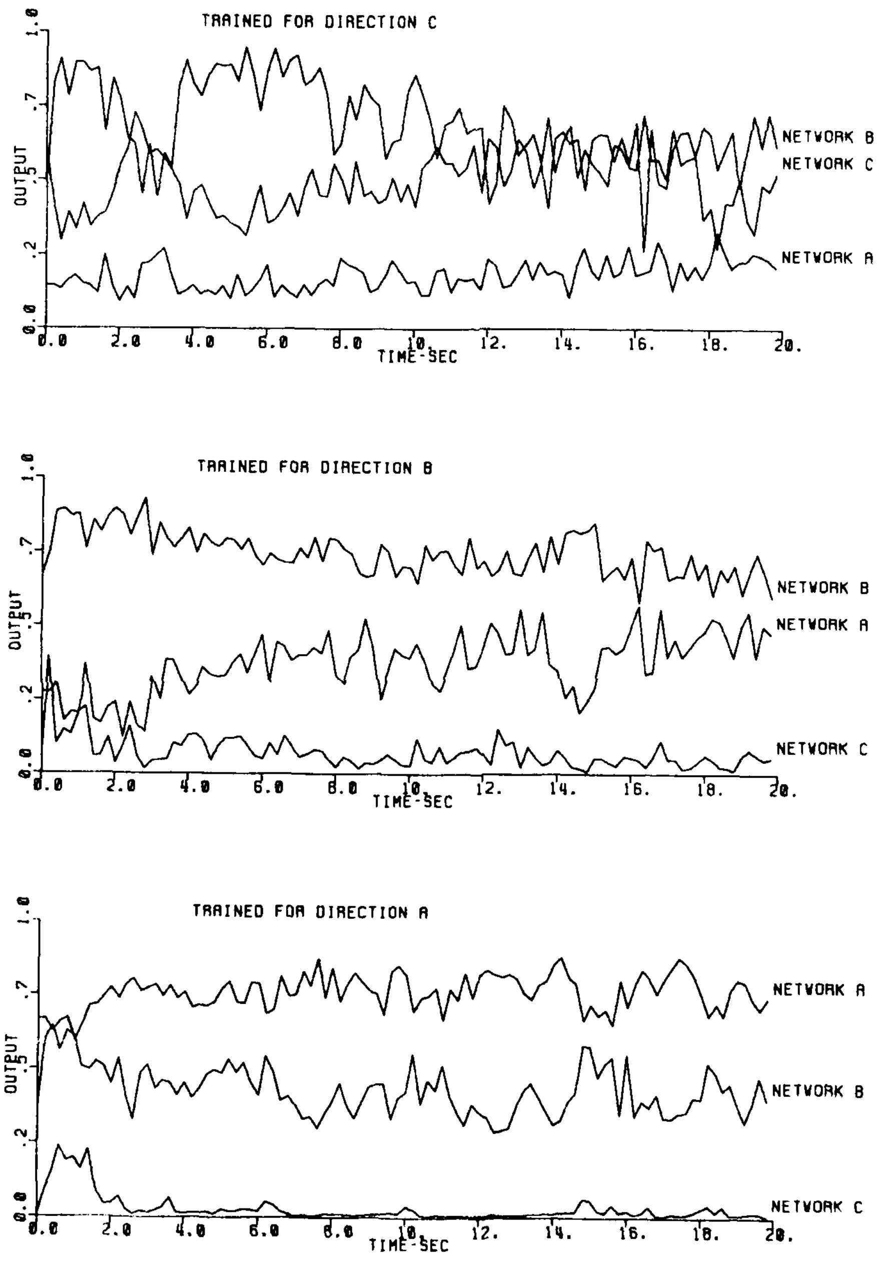


Figure 67

First Simulation with Similar Data as Input

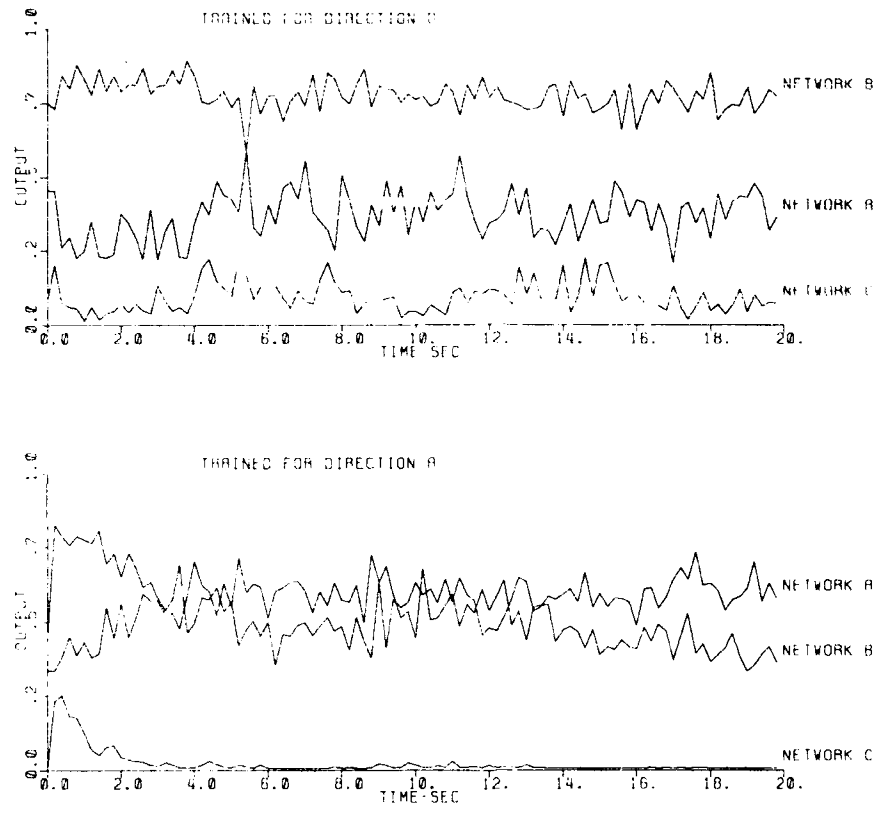
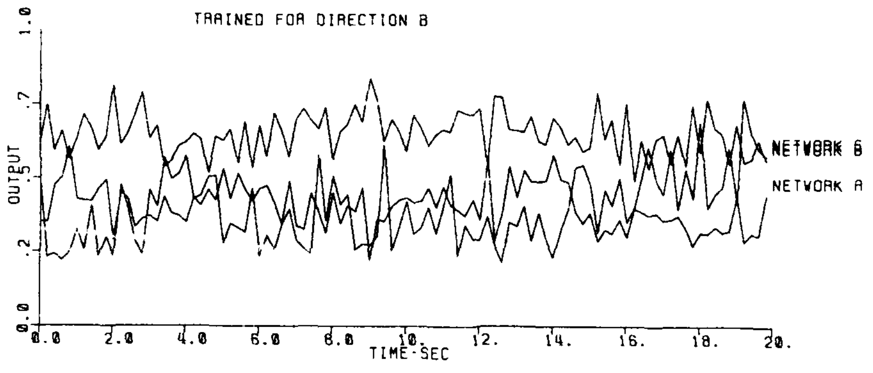


Figure 68

First Simulation with Different Range of Data as Input



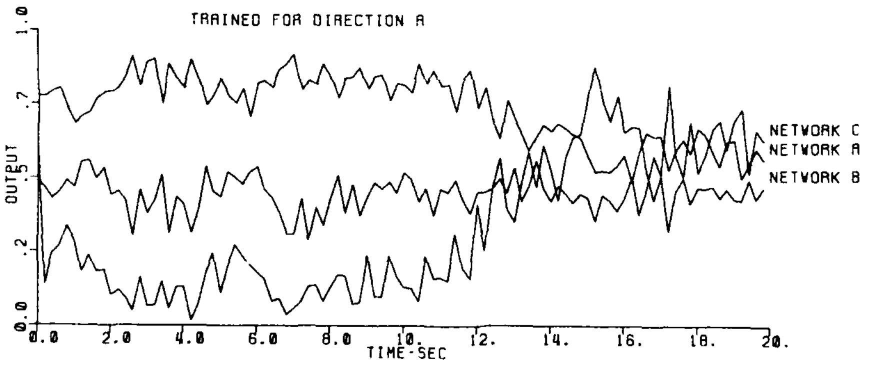
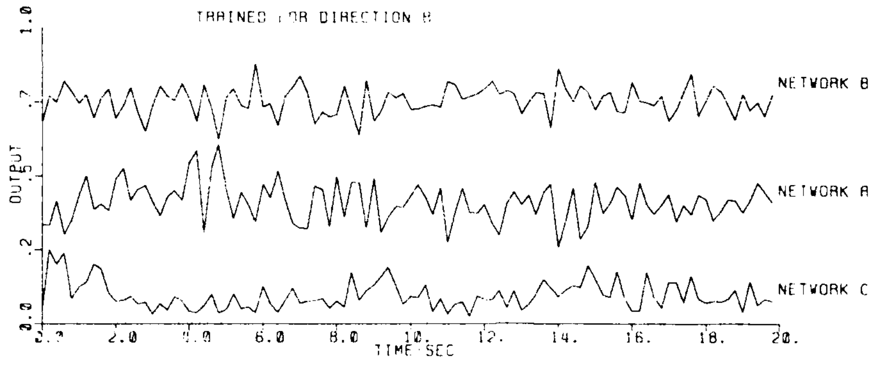


Figure 69

Second Simulation with Different Range of Data as Input



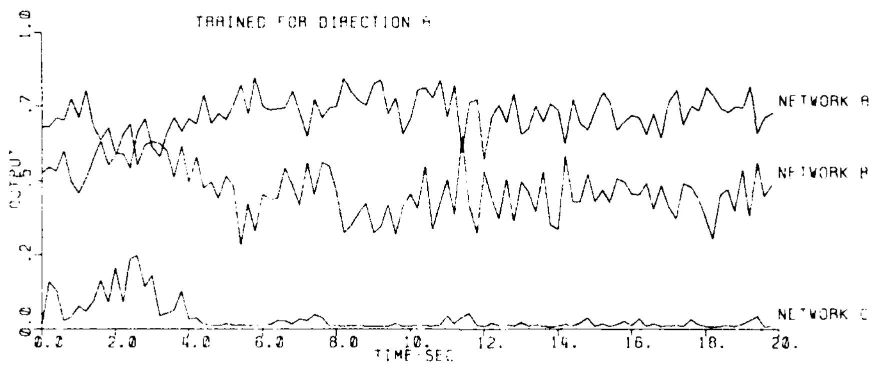
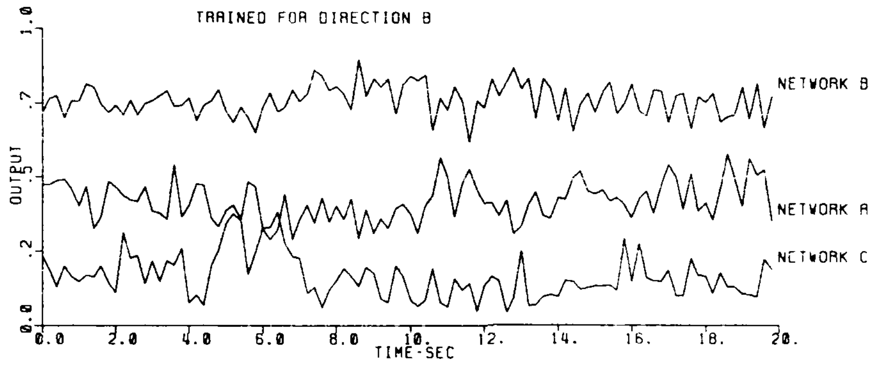


Figure 70

Simulation with Similar Data as Input



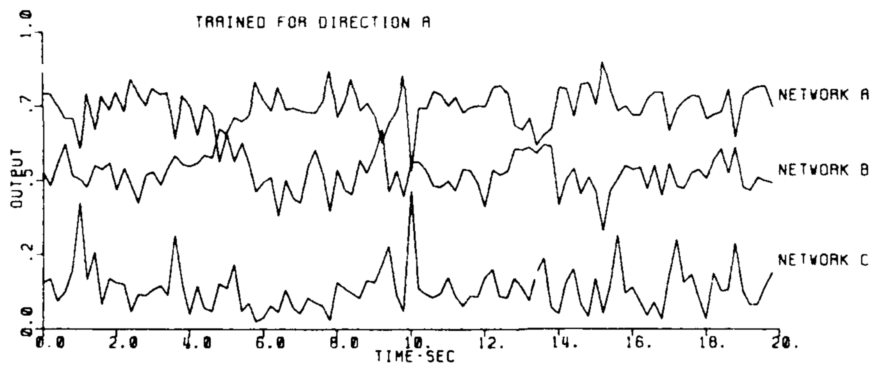


Figure 71

Simulation with Different Range of Data as Input

Test and Range	Direction A	Direction B	Direction C	
Test 1 Range 1	0 %	0 %	0 %	Training Set
Test 2 Range 1	2 %	0 %	27 %	
Test 3 Range 2	31 %	1 %	-	
Test 4 Range 2	17 %	10 %	-	
Test 5 Range 2	5 %	-	-	
Test 6 Range 2	3 %	-	-	
Test 7 Range 3	-	0 %	-	

Table 6

Percent Error in Seismic Test

in the operation of the crane that introduced a type of noise into the system that was not present in the training set. Another important characteristic to notice is that when the signal is sent from direction A or C the distinction from the most distant sensor, C or A, is almost perfect and very distinct. This demonstrates that the three angles could be too close to distinguish with the twelve sensors. The overall success rate for simulations not in the training set was 90.4%. The success rate for the training set was 100%.

The quality of performance of a neural network can best be determined by the networks success with data not contained in the training set. The success rate of 90.4% indicates excellent performance for a system containing large amounts of interference and noise. Correlated noise such as that produced by the crane can cause particularly severe difficulties since the network must be trained to process one signal while ignoring another.

6.3 Comparisons between FFT Beamformers and ANN Beamformers

When comparing FFT beamformers and ANN beamformers several major differences should be considered. These differences are:

(1) FFT beamformers are programmed using DSP hardware and programming languages, while ANN beamformers are taught using modifiable artificial neurons and training algorithms. When the mathematics and programming are simple as is the case with the narrowband beamformer, the FFT method has an advantage. However when the programming becomes more difficult and trade-offs between design complexity and performance are to be made an adaptive system such as ANNs offers advantages.

(2) FFT beamformers require considerably more hardware than ANN beamformers, but since ANNs are trained, training examples must be available for the system to use. Training

can require many iterations for even simple problems.

(3) The steered beams of the FFT beamformer are at arrival angles determined by the formula $\theta = \text{SIN}^{-1}\left(\frac{\alpha\lambda}{2\pi d}\right)$. Since the phases, α, are spaced at equal increments due to the symmetry of the FFT algorithm, the arrival angle beam will be at spacings that are a function of the arcsin of these phases. ANNs, however, can be trained to distinguish any band of angles from any others.

(4) The FFT beamformer is a linear system. The FFT beamformer accepts analog signals digitizes them, processes them and produces analog output signals from a digital to analog converter. As in all linear systems doubling or tripling the input will double or triple the output. The property of superposition will also assure that if two inputs signals are applied, the output will be the sum of the outputs of the two signals applied separately. ANNs however are not linear systems. They accept analog inputs but have outputs that approximate discrete digital levels.

(5) ANNs can be taught to ignore correlated interference that may be present in the environment. FFT beamformers will report correlated interference as a received signal.

(6) FFT beamformers provide an output that varies with time. Since the information is returned to the time domain in the last stage of the beamformer a time history is provided to indicate both the direction of the wave and its signature

as well. The ANN beamformer provides only the directional information.

There are also several similarities between the two beamformers:

(1) The performance degrades as SNR is decreased.

(2) The performance is poorer for wideband signals compared to narrowband signals.

(3) The performance can be improved by increasing the number of sensors and the number of processing elements.

6.3.1 Simulation Comparisons of FFT and ANN Beamformers

In order to compare FFT and ANN beamformers the following simulations were made:

(1) Narrowband beamformers were simulated. Both beamformers used 9 beams. The FFT beams were at equal phase angles and the ANN beams were at equal arrival angles. A narrow amplitude band of .95 to 1.05 was used, and 18 sensors were used. A simulation with no noise and one with a SNR of 7.78 db was made. 20 time samples for the FFT beamformer were used and 20 samples were averaged with the ANN beamformer. (A DFT was actually used since the number of samples was not a power of 2.) The ANN beamformer used four neurons in an elliptical network. The results are shown in Figure 72 through 75.

(2) Wideband beamformers were simulated. The FFT beamformer used 3, 4, and 20 sub-arrays of 18 sensors each. The ANN used multiple arrays in 2, 3, and 4 lines. Each line contained 18 sensors. The spacing for each sub-array of the FFT beamformer was tuned to a frequency half way between the maximum and minimum frequencies for that array. The spacing for each line of the ANN beamformer was set equal to one half the wavelength of the highest frequency to be received. The ANN beamformers were simulated using 20x5 hidden units in a linear style network. Simulations with no noise and simulations with SNRs of 7.78 db were made for both ANN and FFT beamformers. A frequency range of 30 to 300 Hz was used. The results of several of these simulations are shown in Figures 76-83.

In order to compare the results of the two beamformers the outputs were evaluated by comparing their outputs to a specific threshold. Outputs greater than the threshold were considered true and outputs less that the threshold were considered false. For the FFT beamformer a threshold of 0.5 (3 db) was used. The ANN beamformer used 0.7 for the true threshold and 0.2 for the false threshold. Twenty points were averaged on the ANN beamformer to produce the average output. The absolute maximum of the 20 sample time chip was used for the output of the FFT beamformer.

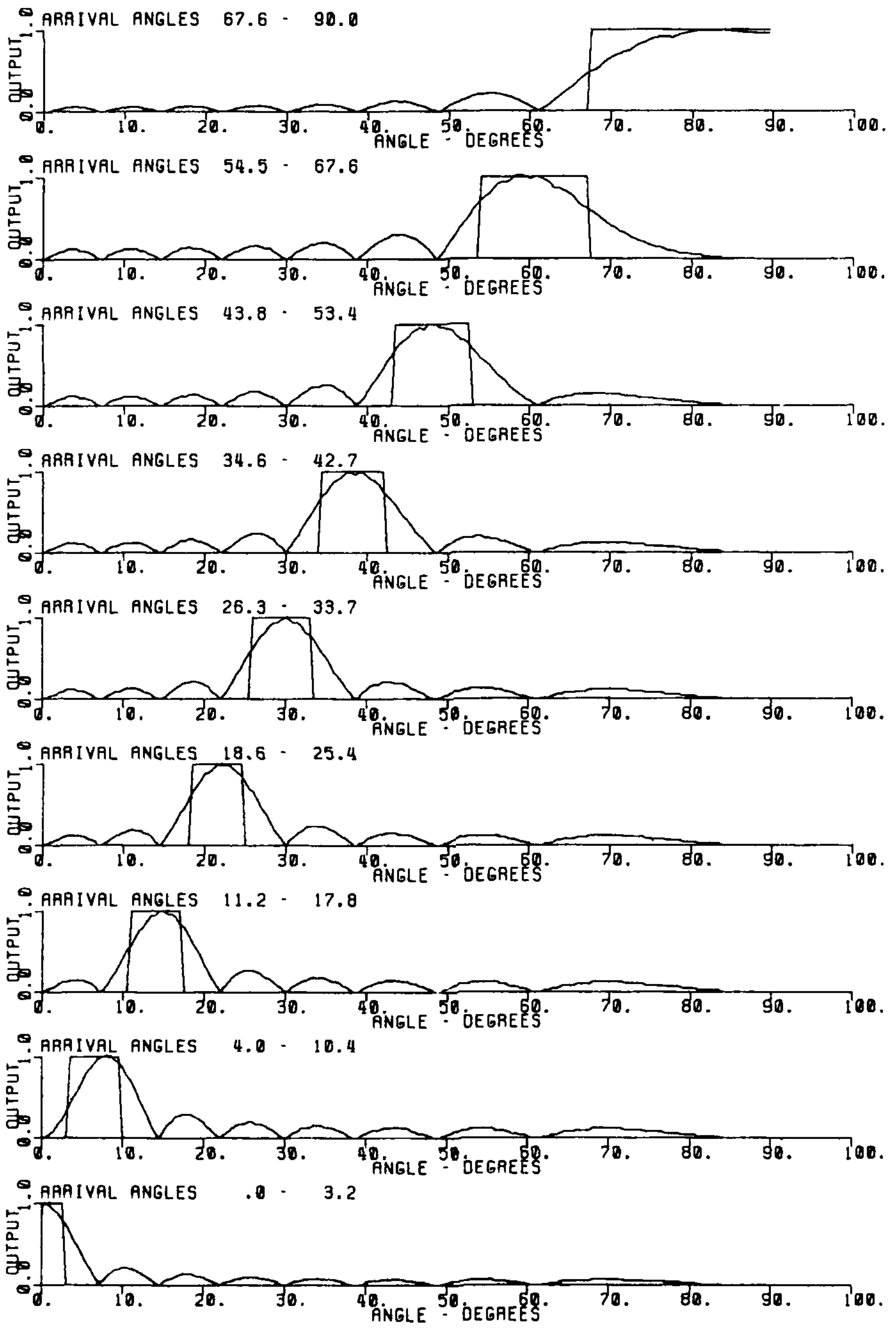


Figure 72

Narrowband DFT Beamformer; No Noise

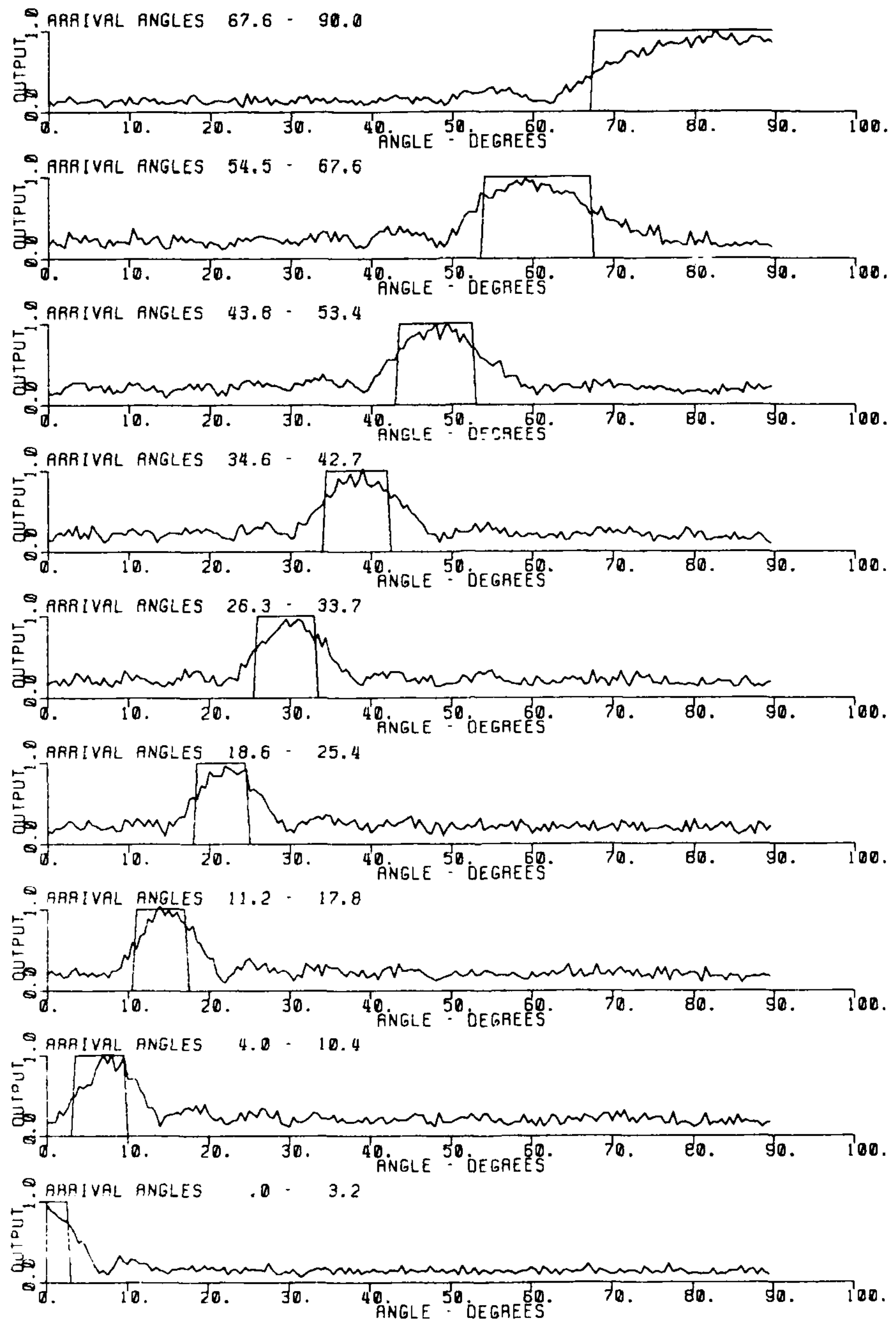


Figure 73

Narrowband DFT Beamformer; Noise Level 1.0

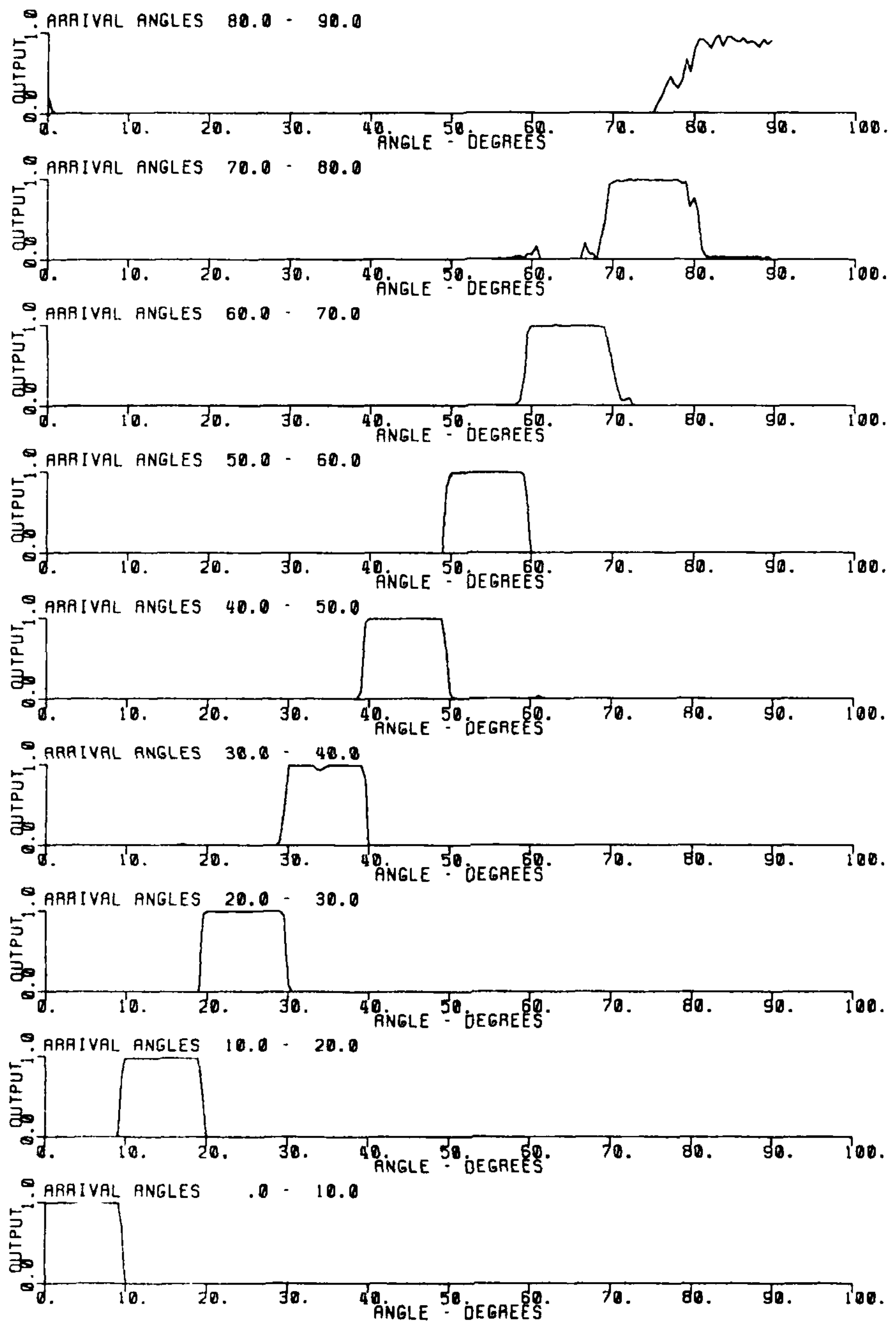


Figure 74

Narrowband ANN Beamformer; No Noise

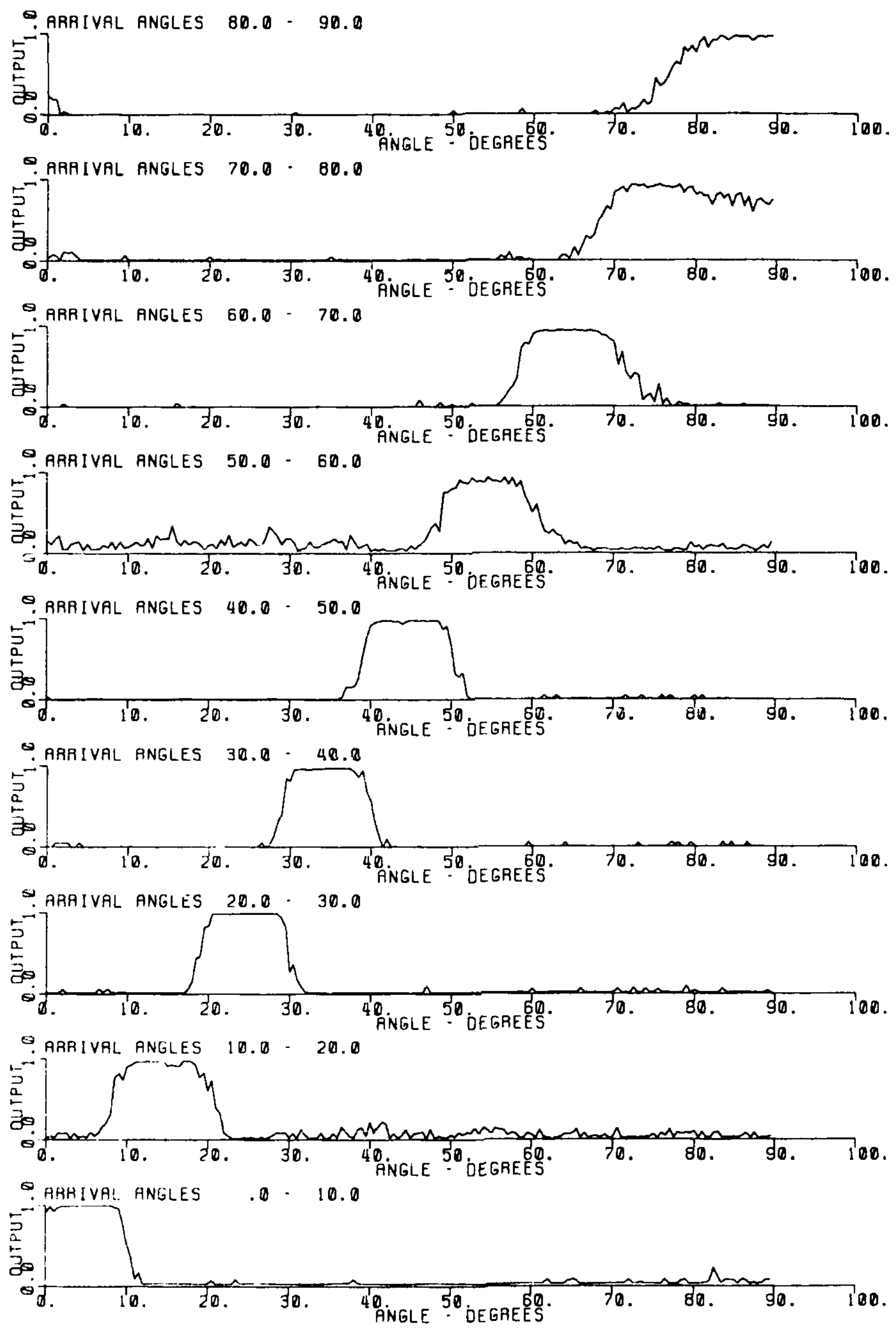


Figure 75

Narrowband ANN Beamformer; Noise Level 1.0

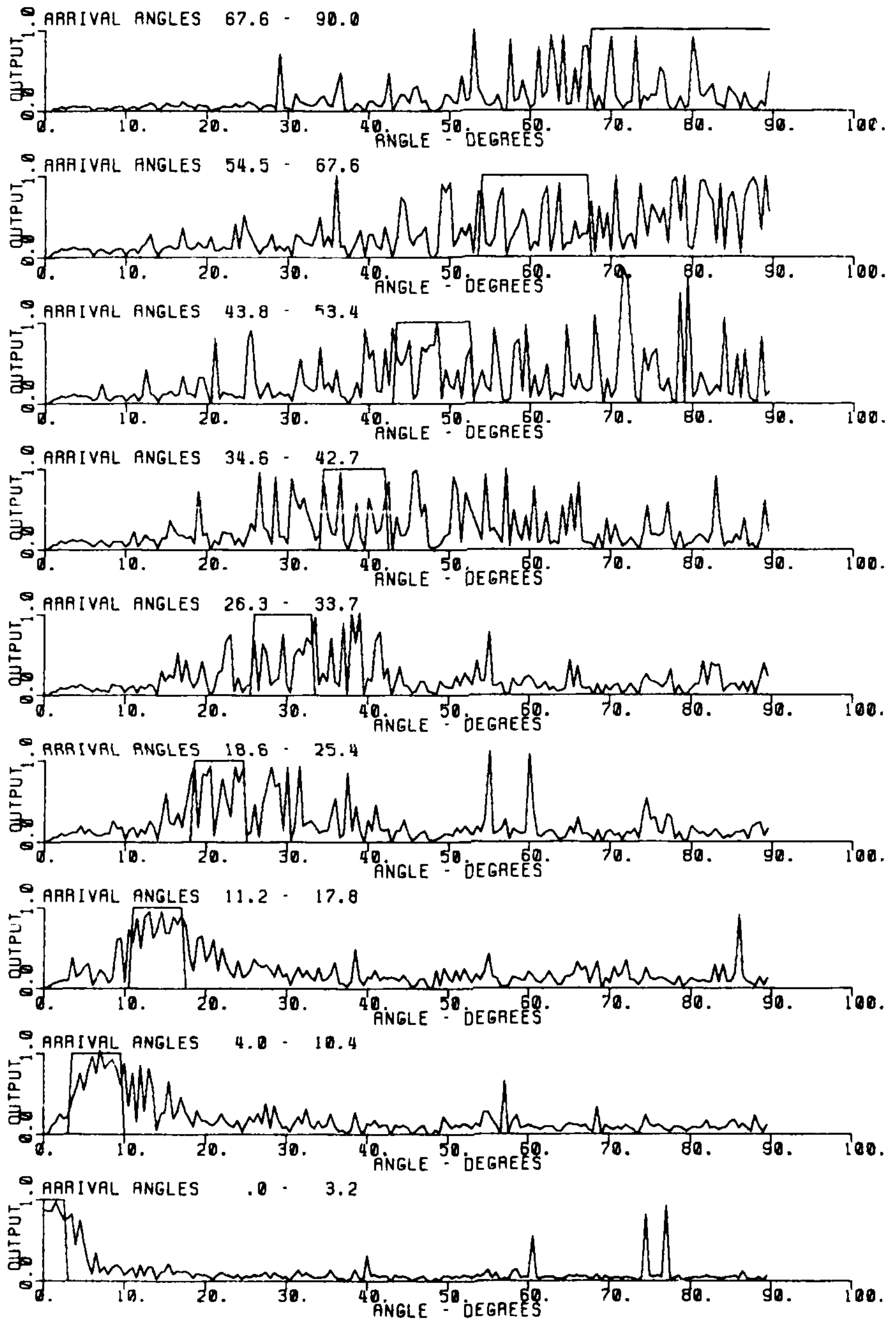


Figure 76

Wideband DFT Beamformer; 3 Sub-arrays; No Noise

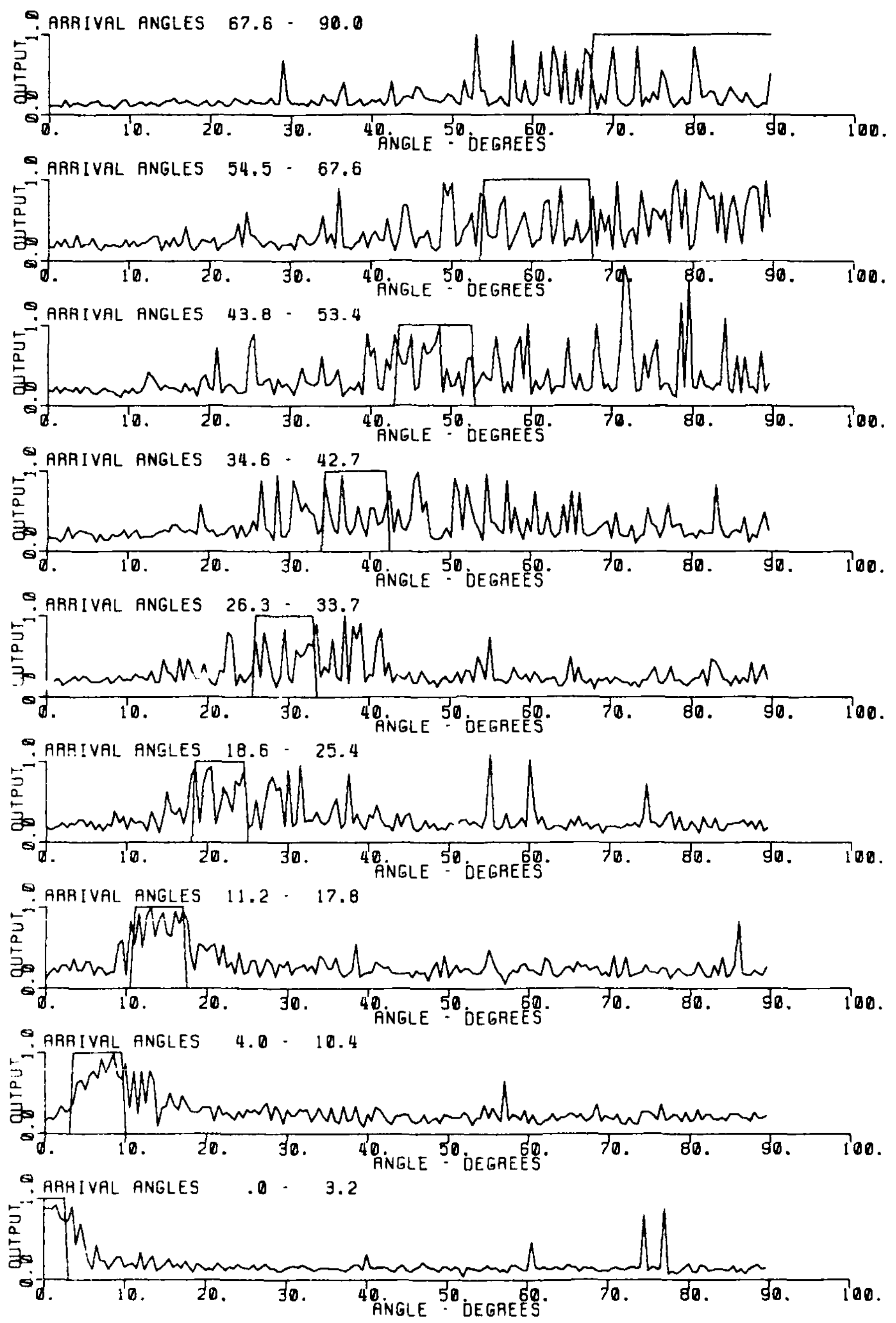


Figure 77

Wideband DFT Beamformer; 3 Sub-arrays; Noise Level 1.0

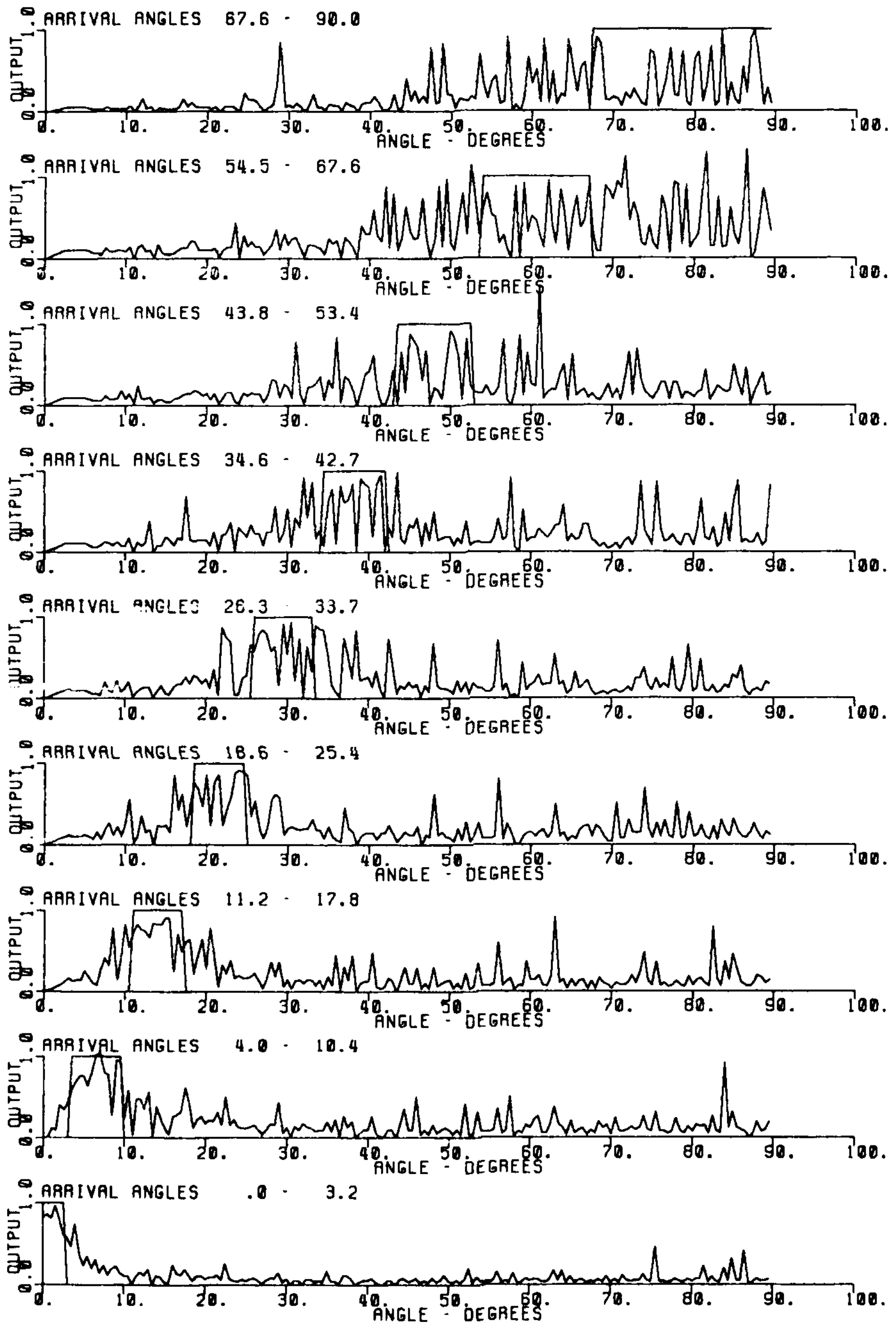


Figure 78

Wideband DFT Beamformer; 4 Sub-arrays; No Noise

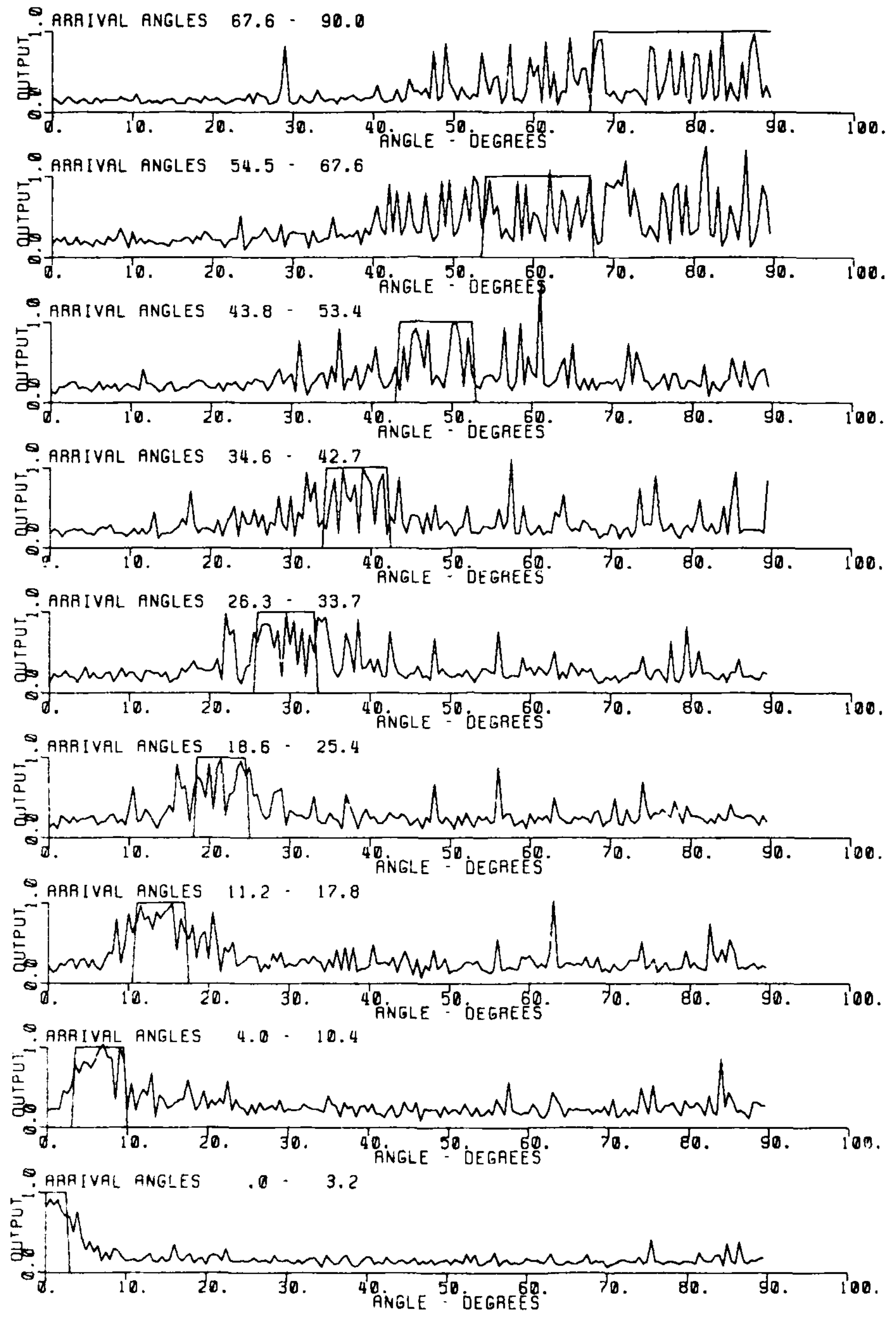


Figure 79

Wideband DFT Beamformer; 4 Sub-arrays; Noise Level 1.0

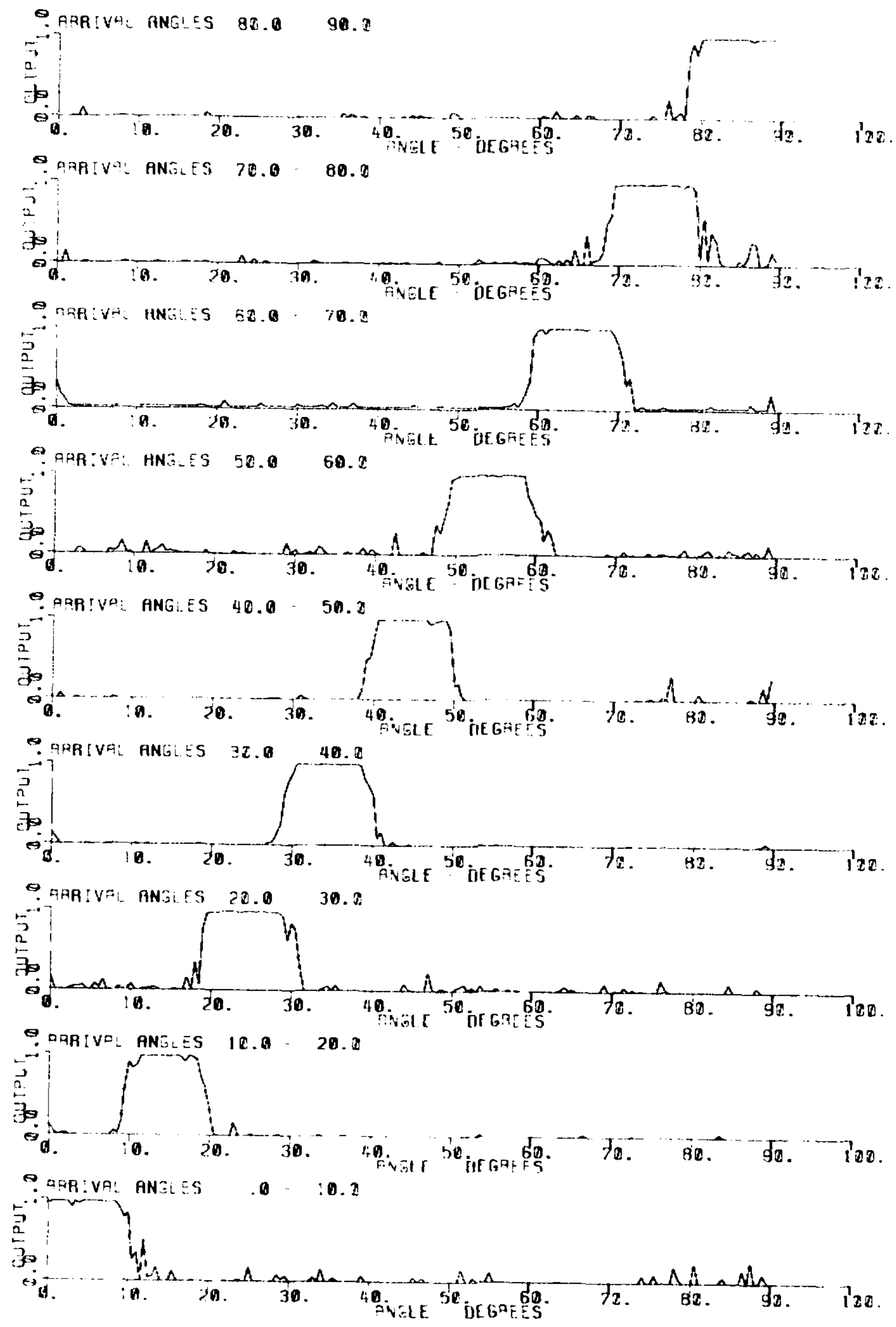


Figure 80

Wideband ANN Beamformer; 2 Arrays; No Noise

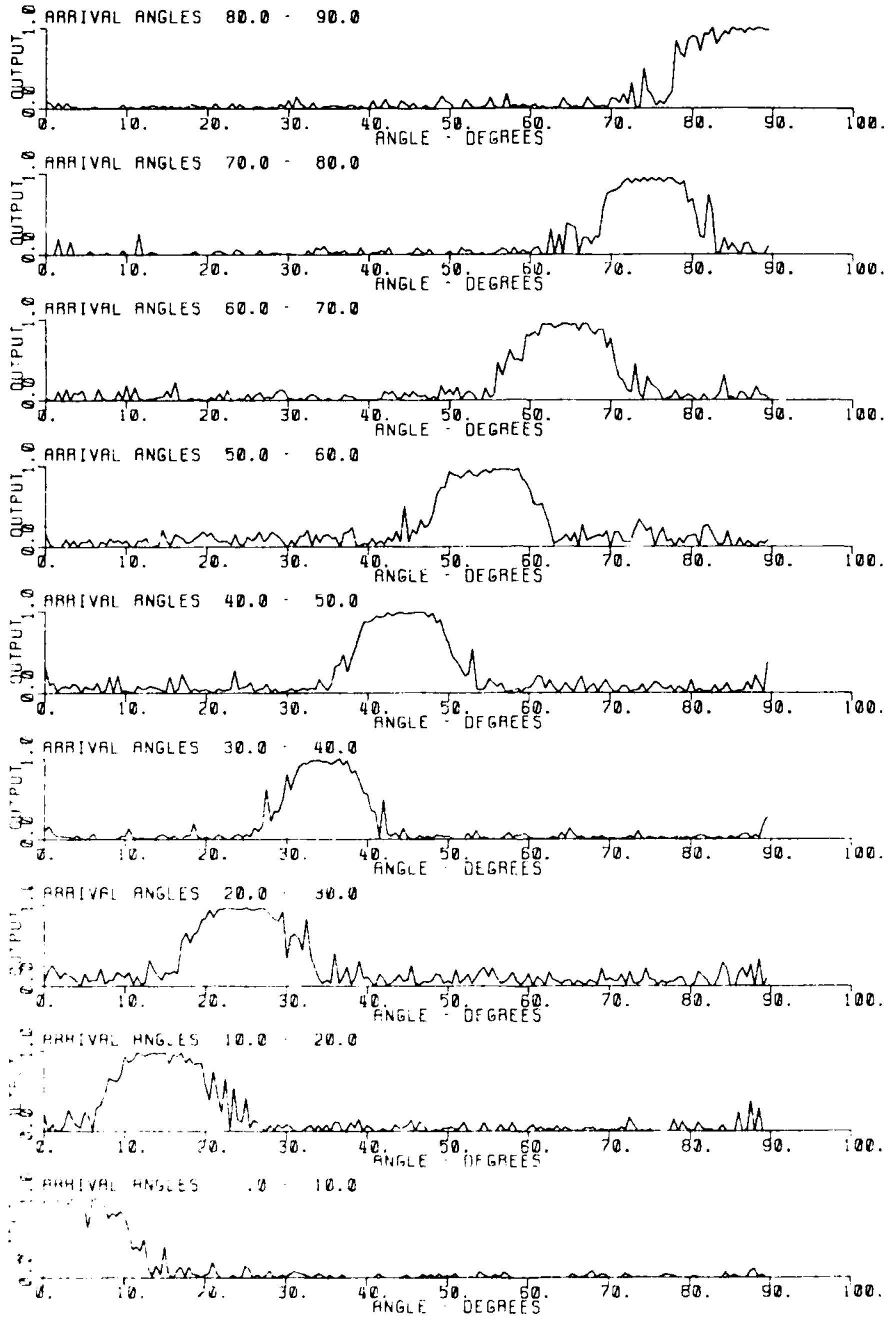


Figure 81

Wideband ANN Beamformer; 2 Arrays; Noise Level 1.0

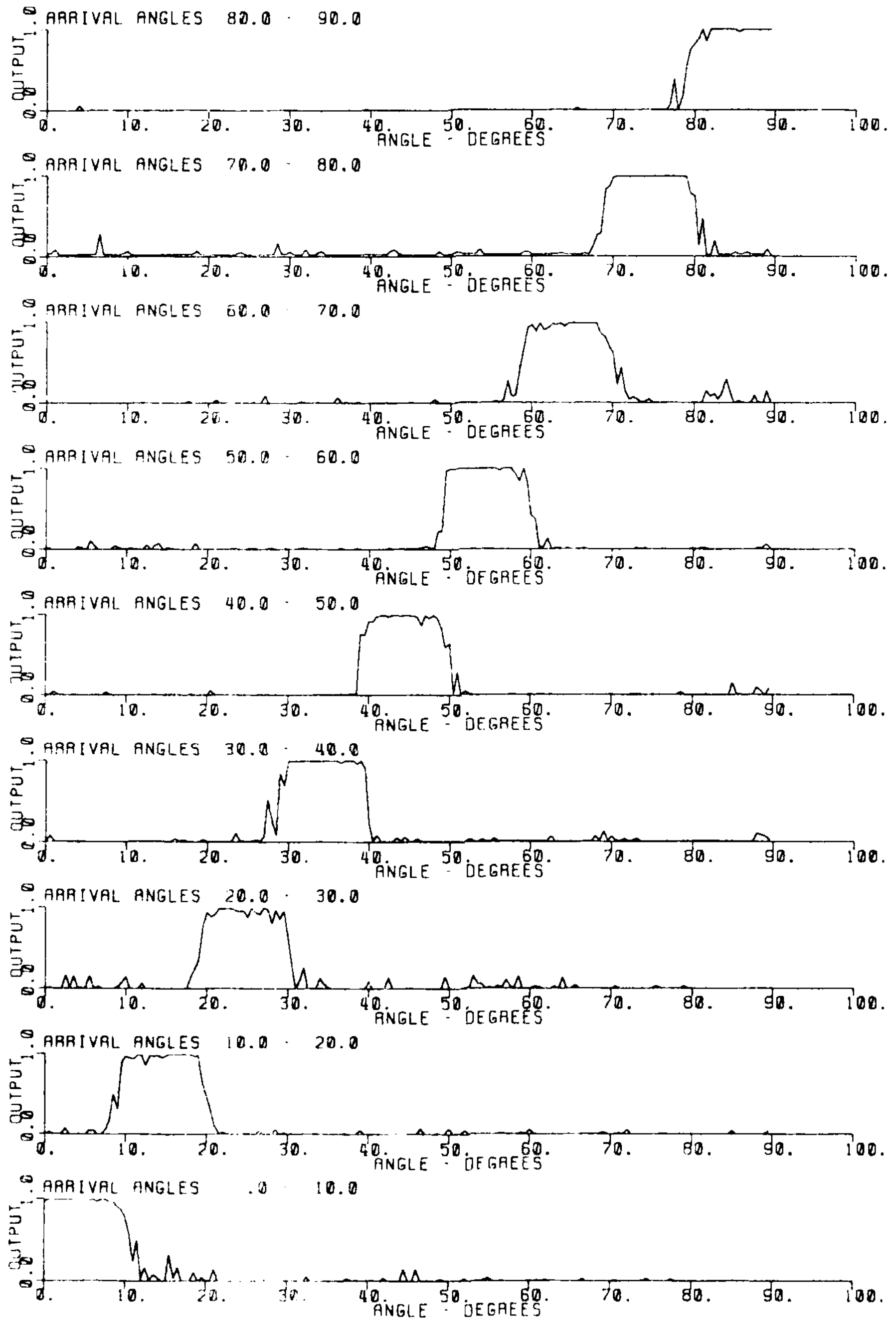


Figure 82

Wideband ANN Beamformer; 3 Arrays; No Noise

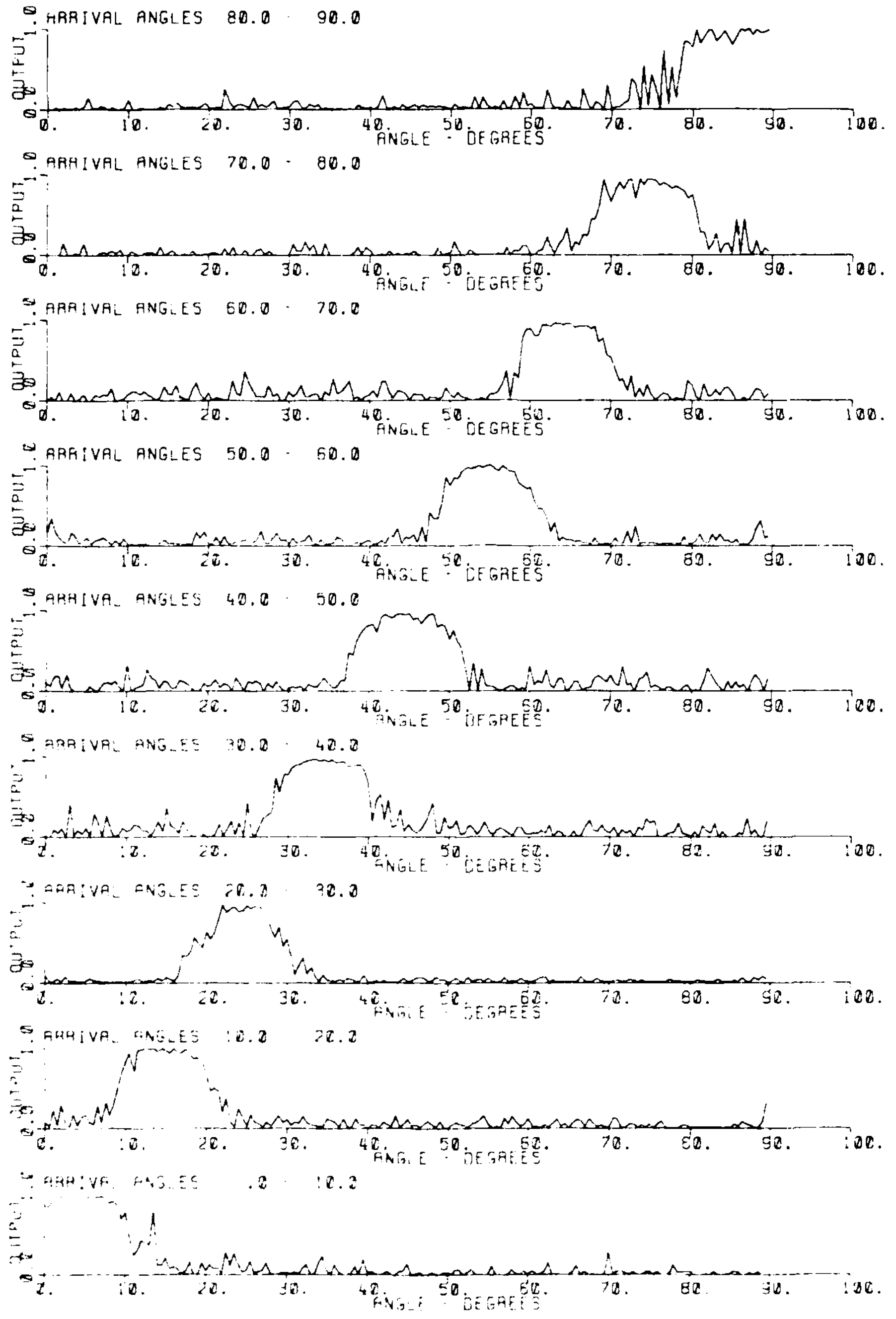


Figure 83

Wideband ANN Beamformer; 3 Arrays; Noise Level 1.0

6.3.2 Results of the Comparison of ANN and FFT Beamformers

The results indicate that both systems provide excellent results for narrowband beamformers. The percent of correct results were 98.5% for the ANN beamformer and 97.2% for the FFT beamformer. Both systems provide good immunity to noise. With an SNR of 7.78 db the ANN beamformers gave 94.3% correct results and the FFT beamformer gave 96.72% correct results.

The wideband results indicate that when a small number of sub-arrays are used the ANN beamformer gives superior results to the FFT beamformer. The results from the simulations indicate that when 3 sub-arrays are used the ANN beamformer gives 96.47% correct results (88.73% correct with 7.78 db SNR noise) and the FFT beamformer gives only 85.11% correct results (84.87% correct with 7.78 db SNR noise). However the results also indicate that the response of the FFT beamformer greatly improve as the number of sub-arrays is increased. When the number of sub-arrays is increased to 20 the performance is improved to give 92.09% correct results (91.10% correct with 7.78 db SRN). However the FFT's response leveled out when at least 2 arrays were used. Each array contained 18 sensors. When more sensors are added no more improvement is realized. This limit in performance was caused by the finite size of the network. It was shown in Chapter 2 that any pattern can be separated from another if enough neurons are used in a three level network. However, when a finite size network is used, only a finite number of separable regions can be distinguished. Additional limitations could be due to the presence of the noise present in the training set.

6.3.3 Processing Requirements for FFT and ANN Beamformers

The amount of time required to process data with an ANN beamformer or an FFT beamformer will be dependent on the exact implementation being used. Systems which employ large amounts of parallelism will give much faster results that pure sequential implementations. The processing requirements for these beamformers can be measured approximately by the number of addition-multiplication operations required to complete the analysis of one cycle of the acquired data. In addition to addition-multiplication operations a non-liner function calculation will be required for each neuron of an ANN beamformer and SIN and COS calculations or fetches will be required for the FFT beamformer. Since these calculations are different for the two beamformers, the following comparison must be considered to be only approximate.

The number of addition-multiplication operations for an ANN beamformer can be calculated as follows:

$$G*N*(S*H1+H1*H2+H2*H3). \qquad (6.1)$$

Where N is the number of Networks (one for each angle bin).

S is the number of input sensors.

G is the number of iterations averaged.

H1 is the number of hidden units in level 1.

H2 is the number of hidden units in level 2.

H3 is the number of hidden units in level 3.

The number of addition-multiplication operations for an FFT beamformer can be calculated as follows:

$$B*((1+K)*S*T*\log 2(T)+T*S*\log 2(S)). \qquad (6.2)$$

Where B is the number of addition-multiplication operations required for one complex butterfly operation. (B=4 will be used for this comparison).

K is the number of sub-arrays.

S is the number of input sensors.

T is the number of time samples.

Using these formulas the following comparison can be made:

Narrowband ANN beamformer with 9 angle bins,

20 iterations averaged.

18 sensors

4 hidden units on level 1

1 hidden unit on level 2

Addition-multiplication operations = 13,680

Narrowband FFT beamformer with 9 angle bins,

18 sensors (1 sub-array)

20 time samples

Addition-multiplication operations = 18,446

Several items should be noted about these comparisons.

(1) The FFT is designed to be used with a power of two.

The 20 time samples and 18 sensors were used for comparison with the ANN beamformer.

(2) A complex FFT was used in the above calculations. When real data is used, modifications to the FFT algorithm can be made that improve speed by nearly 50%.

(3) No allowance was made for the SIN and COS calculations in the FFT beamformer or the nonlinear function calculations in the ANN beamformer.

These results indicate that for small narrowband arrays the processing requirements of the two systems are very similar. However, as more and more sub-arrays are required for processing FFT wideband beamformers, the number of addition-multiplication operations will increase proportionately.

6.3.4 Summary of FFT and ANN Beamformer Comparison

Throughout this dissertation the response of ANN beamformers to several design and input characteristics has been noted. The following itemized list is a comparison between ANN and FFT beamformers for these characteristics.

Design variables for ANN Beamformers:

(1) Number of Hidden Units.

No further improvement in the response of an ANN beamformer is achieved when the number of hidden units is increased past approximately 4 for narrowband beamformers with preprocessing or approximately 20 for wideband beamformers.

(2) Number of Sensors.

No further improvement is the response of an ANN beamformer is achieved when the number of sensors is increased past approximately 36. The sensors for wideband ANN beamformers must be in two different directions, preferably at 90 degree angles. Two dimensional beamformers improve the response of narrowband beamformers as well.

(3) Preprocessing.

When narrowband signals are used providing the square of the sensor output to the network improves the response considerably. A network with this modification to the inputs can produce results with 4 hidden units that would require roughly 20 hidden units when used with non-preprocessed inputs.

Design variables for FFT Beamformers:

The main design variables for the FFT beamformer are the number of sensors, the number of time samples used in each time chip, the number of sub-arrays, the sampling rate and the gage placement distance. These variables can be used to produce the desired level of response, but when wideband signals are to be detected many sub-arrays can be required. Details on the design of FFT beamformers can be found in Follett [5].

Input Variables:

(1) Signal to Noise Ratio.

ANN beamformers provide very good immunity to noise, but FFT beamformers are much better. Increasing the noise from no noise to an SNR of 7.78 db will change the accuracy of an ANN beamformer from 96.47% to 88.73%, but the FFT beamformer will change from 85.11% to 84.87%. Even though the performance of this FFT beamformer was inferior to that of the ANN beamformer, the effect of adding noise was much more harmful to the ANN beamformer.

(2) Amplitude Changes.

ANN beamformers can be trained to be very tolerant of amplitude changes, but examples of all amplitudes to be detected must be included in the training set. FFT beamformers are designed to operate at a given amplitude. If the input is decreased the output will be decreased proportionately.

(3) Variations in Bandwidth.

ANN beamformers can be trained to be tolerant of wideband signals. Three levels of perceptrons will be required due to the complexity of the patterns to be distinguished. FFT beamformers are not as tolerant of wideband signals. Designs must be modified to include many sub-arrays in order to furnish all of the tuned frequencies needed to analyze the wideband of frequencies. One general rule suggested by Follett [5] is to only allow one sub-array's bandwidth to expand 20% of the tuned frequency.

Other Variables:

(1) Dependency on Time.

Both ANN and FFT beamformers are very dependent on time.

The number of samples used by an FFT beamformer is one of the design variables that determines the size of the first and last level of FFTs. Time is also very important in the ANN beamformer. As was shown in Figures 50 and 51, the network produces many outputs over the course of a cycle that are erroneous. Time averaging greatly reduces these errors. The period over which the average is taken should be at least that of the lowest frequency being received by the beamformer.

(2) Training Methods.

Backpropagation is the best training method for training ANNs studied in this work. It can train networks with any level of perceptrons. It can also train a network to ignore unwanted signals and adapt to the environment in which it is trained. FFT beamformers must be programmed rather than trained.

Chapter 7

Proposed Design Criteria

7.1 Summary of Beamformer Design Criteria

It has been demonstrated in the preceeding chapters that ANNs can be used successfully to determine the angle of arrival of a wavefront using a beamformer array. Either narrowband or wideband systems can be implemented. When designing a feedforward neural network to process information from an array of sensors the main considerations are:

(1) The number of sensors used.

(2) Then configuration of the sensors.

(3) The number of neurons in the network.

(4) The configuration of the network.

The sensor configuration must be two dimensional and the ANN must have three levels if wideband signals are to be processed. Narrowband signals can be processed with one dimensional arrays and only two level ANNs. However two dimensional arrays will improve the response of a narrowband signal especially in the higher angle bands.

It was demonstrated in previous chapters that the successfulness of these networks varies directly with the number of sensors, the number of hidden units, and inversely with the noise level, the amplitude range, and the frequency range. To demonstrate the degree of effect that these variables have on the network five plots are presented that show the result of additional simulations that were performed. These plots are shown in Figures 84 through 88. In these simulations each output was evaluated by the following criteria. If the target value was 1.0 and the result was greater than 0.7, the result is considered correct. If the target value was 0.0 and the result was less than 0.2, the result was considered correct. Results between 0.2 and 0.7 were considered erroneous. The average of correct results of all angle bands was calculated for the following plots. These plots are presented only to demonstrate the trend that is caused by the variable in question. An accurate evaluation should be made by examining the simulation outputs generated by the program VWBBFNN in Appendix I.

It should be noted that the desired output for each of the 9 simulation bands is zero for 89% of the arrival angles and one for 11% of the arrival angles. Sometimes a network will converge to a local minimum which produces a constant result of zero, one or 0.5 for all angle bands. It is important to view the output listings of these simulations to ensure that one of these erroneous local minima has not been reached.

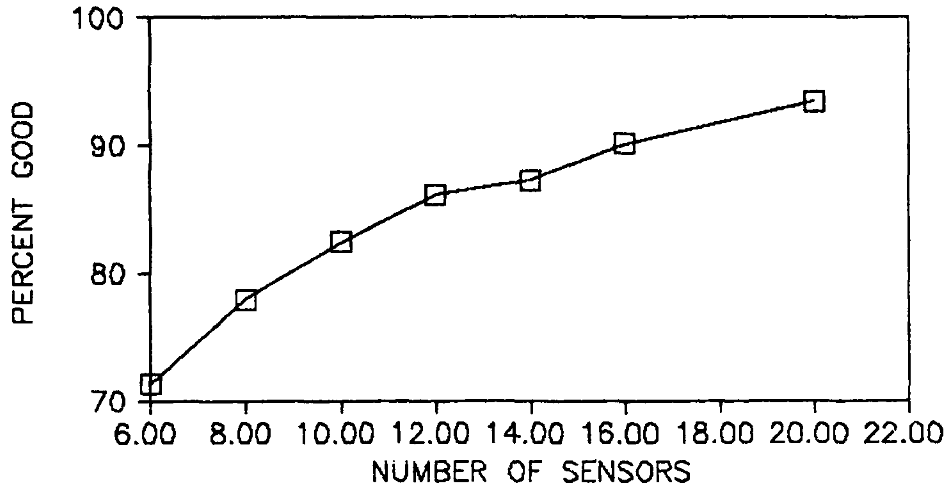


Figure 84

Number of Sensors Versus % Good

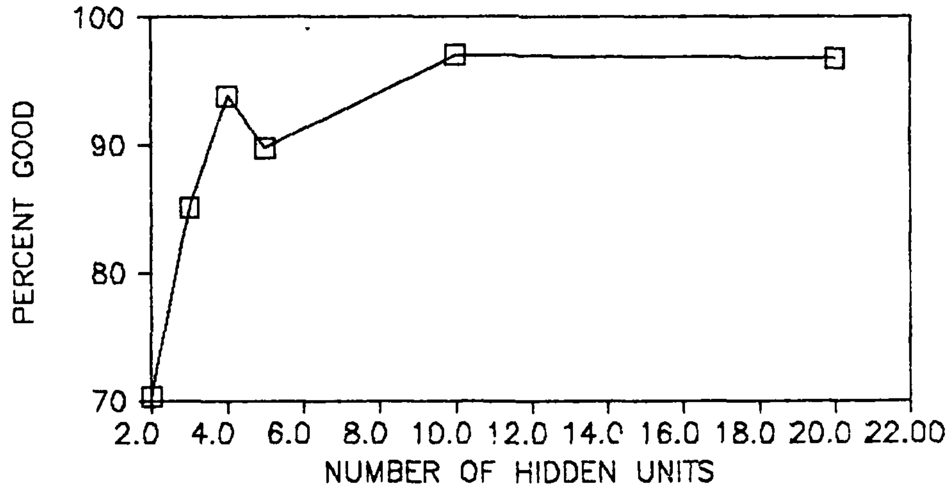


Figure 85

Number of Hidden Units Versus % Good

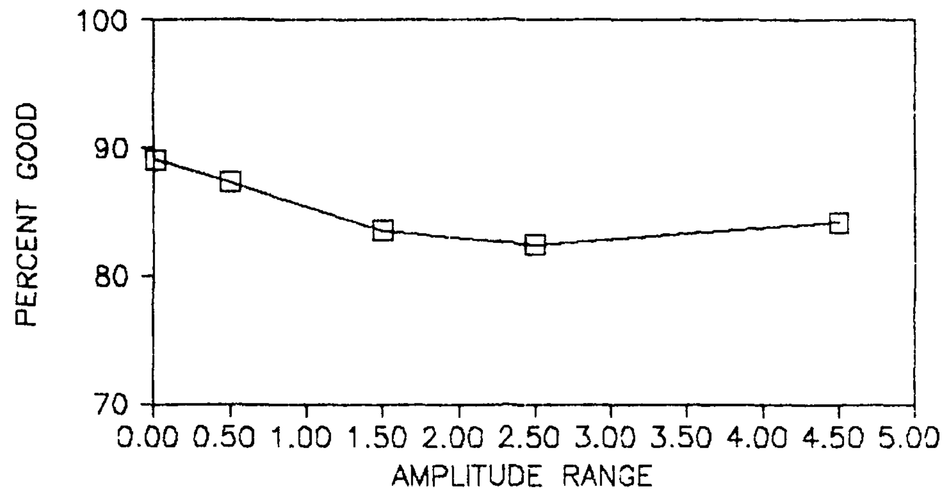


Figure 86

Amplitude Range Versus % Good

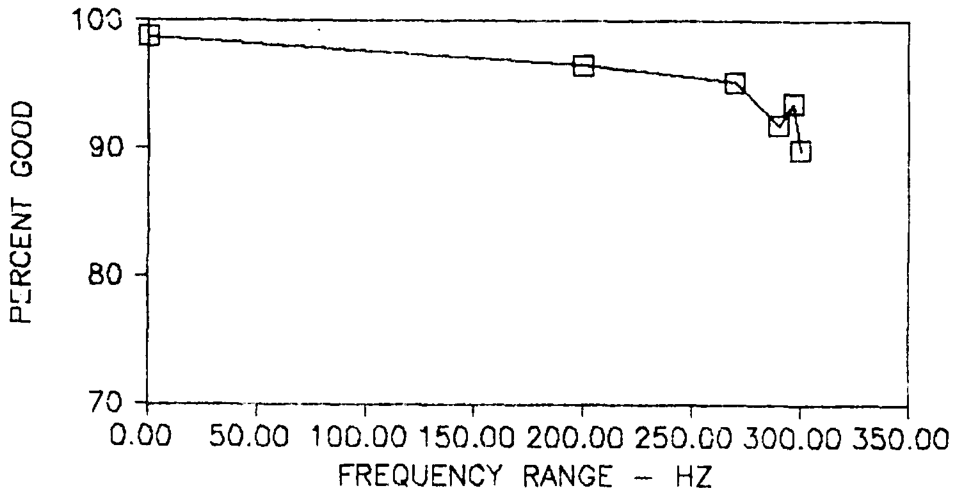


Figure 87

Frequency Range Versus % Good

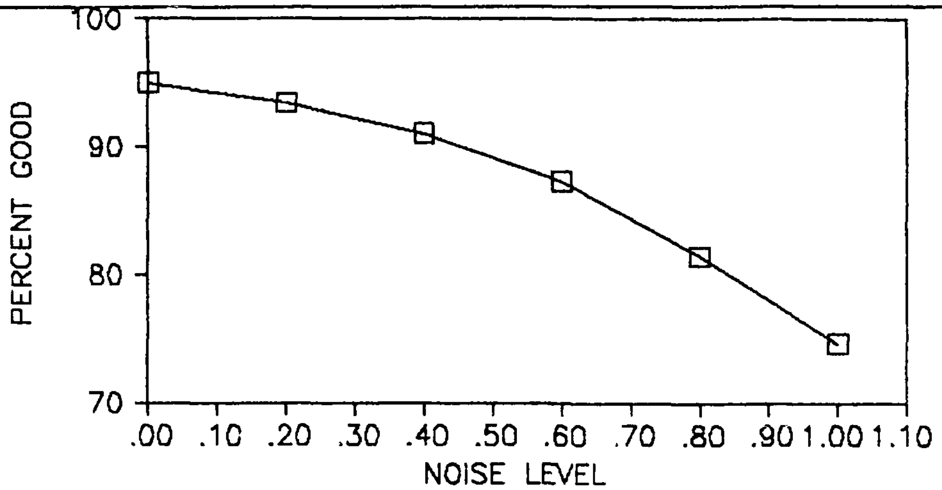


Figure 88

Noise Level Versus % Good

For each of these plots all variables remained constant except the one being tested. The program VWBBFNN in Appendix I can be used to simulate any combination of these variables. By using this program various simulations can be processed to determine the best simulation for a potential problem.

7.2 Description of Simulation Program

The Variable Wideband Beamformer Neural Network program VWBBFNN was designed to train and simulate feedforward neural networks connected to beamformer arrays. The training algorithm used is backpropagation with momentum. Two, three, or four levels of neurons can be used in the simulation. The program must be supplied with the following information:

(1) Sensor locations.

(2) Arrival angles that are to be learned.

(3) Training Rate.

(4) Training momentum.

(5) The number of training iterations.

(6) The number of hidden units on each level.

(7) The number of frequency components that make up a wide band signal.

(8) The frequency range to be used.

(9) The zero to peak amplitude range to be used.

(10) The peak to peak noise level to be used.

(11) The type of input preprocessing (if any).

Different training rates and momentum values can be used throughout a training session. The program supplies the user with the learned weights and a simulation of the learned network. A simulation is performed for each angle band that was specified. These simulations are conducted over a range of 0 to 90 degrees at intervals of 0.5 degrees. At each interval twenty random times are chosen and the maximum, minimum and average output values for the network at these ten time samples were calculated and saved. The output is scaled as integers between 0 and 100. At the end of a simulation the number of correct, erroneous, and ambiguous results are recorded. The same environmental parameters for frequency range and noise level that were used in the training session are used in the simulation. The training amplitude range is reduced by 10 % for the simulation.

The training session begins by inquiring if the network has already been through previous training sessions. If another training session exist, the weights are read into memory; if not, the weights are preset to small random values. When training starts the input patterns are presented at random in the following way. For each iteration a sample of each angle band is presented once except for the two on either side of the target angle band. These two are presented twice. The pattern within the target band is presented N times where N is the number of angle bands in the network. For example if nine bands were being used and the network is being trained to detect angles within band 4, the patterns in band 4 would be presented nine times, the patterns within band 5 and 3 would be presented two times and bands 1,2,6,7,8 and 9 would be presented once each. Many modifications to this method were attempted and this method worked best to emphasize training at the locations where it is required the most.

The instructions are supplied to the program in three ASCII files. The first file is named "WB.INS" and contains the training and simulation parameters discussed above. In addition to these parameters the names of the other two input files and a base name for an output file are included. These two additional files contain the sensor's X and Y coordinates and the angle ranges for each arrival angle to be learned. The output files are labeled with the base output name and the extension ".LOG" for the simulation log and the extension ".Wnn" on each of the weight files. ("nn" is the number of the band for each angle).

7.3 Examples of Beamformer Design

Consider the sample problem.

A wideband beamformer is to be designed with a frequency range of 100 to 300 HZ, an amplitude range of 1. to 3. and a noise level of 0.5. A first attempt was made by instructing the program to use ten sensors in a straight line configuration. The result of this simulation is shown in Figure 89. The response indicates that the 100 to 300 Hz bandwidth is too wide to be processed with a one dimensional array. A second simulation was made with a two dimensional array of ten sensors in each direction. This simulation is shown in Figure 90. These results showed a tremendous improvement. A third simulation was made with 30 sensors in which a third row was placed at a 45 degree angle with the other two sensor lines. The results of this simulation are shown in Figure 91. These results showed little improvement in the response of the network. This indicates that 20 sensors should be near the optimum for this problem. Further simulations could be made to increase or decrease the number of hidden units to further optimize the beamformer's response to the desired level.

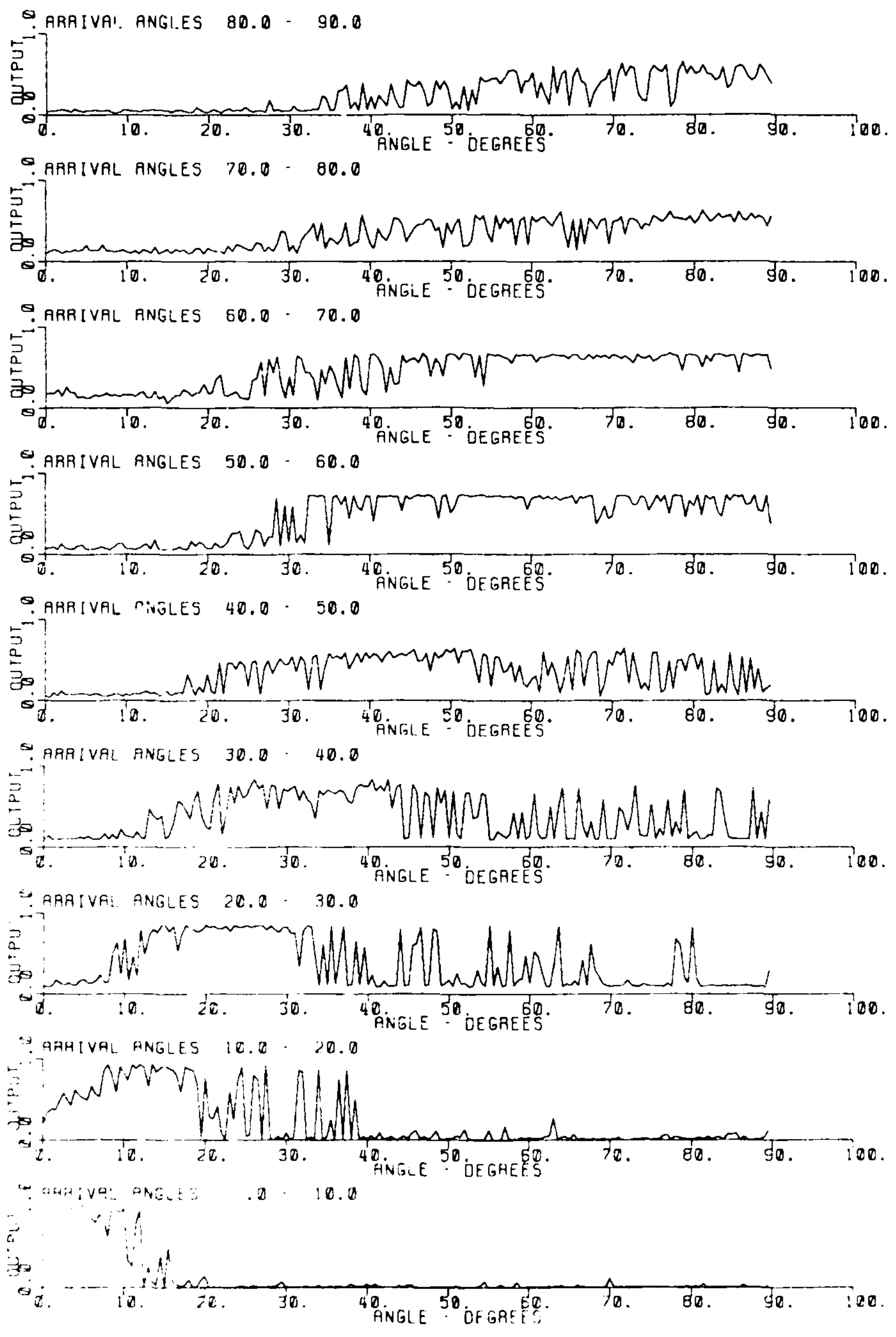


Figure 89

Design Example Simulation with 10 Sensors

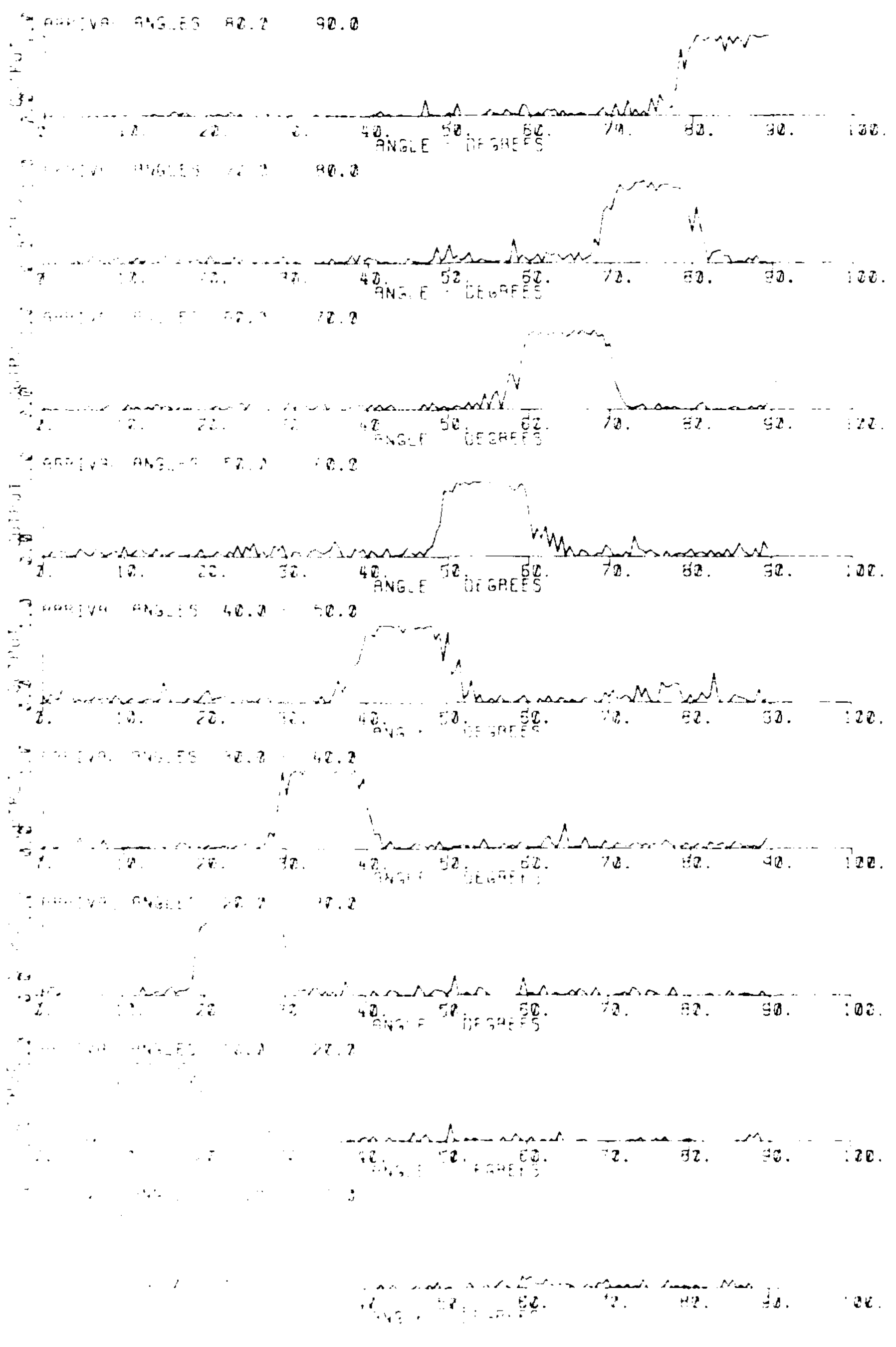


Figure 90

ulation with 20 Sensors

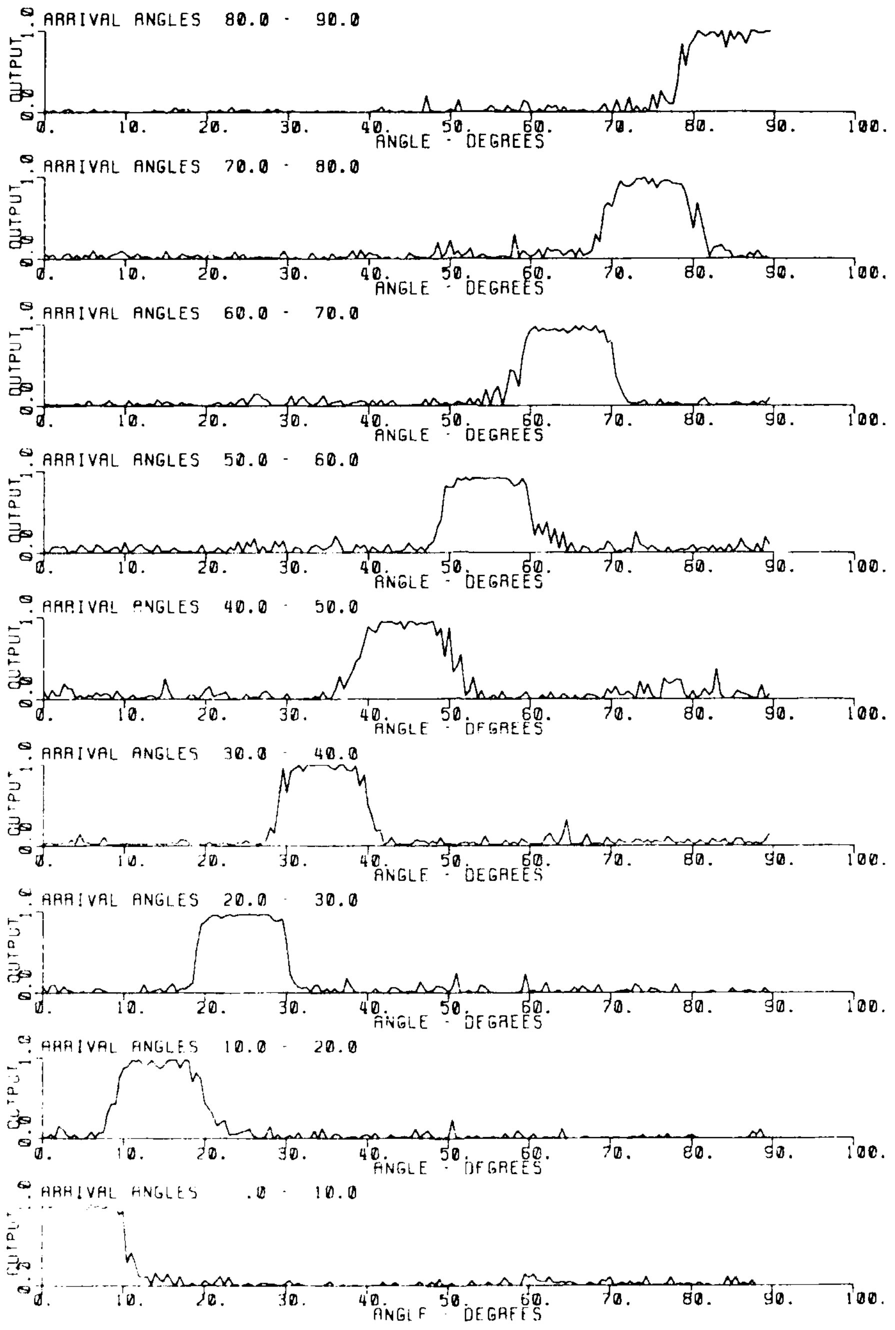


Figure 91

Design Example Simulation with 30 Sensors

Chapter 8

Survey of ANN Hardware

8.1 Artificial Neural Network Hardware

Research in Neural Networks is being performed on a variety of hardware. Conventional computers from PCs to supercomputers are used to process training sessions and output simulations. In addition to these conventional computers several vendors manufacture add-on processors to aid a host computer in processing ANN computations. These add-on processors are usually digital and take the form of a Floating-point Processor.

One of the main hopes for Neural Networks is that they can be produced in a very dense package. Very primitive integrated circuits with only a few analog neurons are currently available. If integrated circuits can be produced which enable networks of 100's or 1000's of neurons to be incorporated in a single package, many difficult cognitive problems can be solved that are currently very difficult to solve on von Neumann Computers. These integrated circuits could be either digital or analog or a combination of the two.

8.2 ANN Integrated Circuits

ANN integrated circuits are divided into two main groups, analog and digital. There is some controversy as to which way the industry should evolve. Digital implementations have the advantage of being deterministic and able to be designed to any degree of accuracy. However the extreme complexity of multipliers, adders, and nonlinear function evaluators make the implementation of large feedforward networks unfeasible in digital technologies. Analog implementations seem to be the most promising for large networks on a single chip. They provide improved speed; they are much more conservative with silicon space; they require fewer pins; and they can consequently be connected in more elaborate architectures. Some of the criteria that should be considered when choosing an implementation of ANNs are:

(1) Does the chip perform on or off chip learning?

(2) What algorithms will the network work with?
(Backpropagation, Hopfield Nets, ART, etc)

(3) Are the interconnections programmable?

(4) Are the interconnections analog or digital and what is their precision?

(5) Can neurons be connected globally?

(6) What is the chip's speed?

(7) What is the price per neuron?

(8) What is the power consumption per neuron?

(9) What is the chip size?

(10) What are the input/output characteristics?

(11) Does the chip guarantee stability?

(12) How easily is the chip reproduced?(13) What is the chip's temperature of operation?

(14) How expandable are networks with this chip?

(Summarized from [22])

High marks on all these criteria describe an ideal chip. These requirements will definitely need to be relaxed. This is especially true for backpropagation algorithms. The complexity that would be required to implement feedforward networks using backpropagation will be especially difficult. These difficulties are due to the high computational requirements required when training networks with backpropagation.

There are several basic problems that have hindered successful large scale integration of ANNs. The most severe are:

(1) The inability to implement variable resistors on an integrated circuit to serve as the weights that emulate the synapses.

(2) The connections problem.

In fully connected networks every neuron in one level is connected to every neuron in the following level. The number of connections that must be routed becomes extremely large for even moderate size networks. Some implementations try to solve this problem by limiting each neuron's connections to its nearest neighbor. Limiting the connections to severely can degrade the network's response.

Some researchers have had limited success with these problems, but most have used switching between fixed resistors or hybrid methods such as multiplying digital to analog converters to accomplish this end.

8.2.1 Available Integrated Circuits

At this time several companies have announced plans to market a neural network chip or have made one commercially available. Three of these are INTEL, Fujitsu, and Syntonics. The Syntonics is commercially available at this time. It is available on an evaluation board known as the DENDROS-1 and has the following specifications [23]:

(1) ART-1 Network Model (Adaptive Resonance Theory, Section 2.2.4)

(2) Connection is to the second nearest neighbor
(the nearest neurons in the following level)

(3) Self-adaptive programmability

(4) On-chip learning

(5) Fully Parallel

(6) Analog implemented in CMOS

(7) Continuous operation with 10 to 20 msec settling time

(8) Synapses are implemented with capacitors

(9) 8 Neurons

(10) There are 58 Synapses

30 Modifiable

5 All-or-none

23 Fixed

This IC is one of the first to become commercially available. It is primitive in that it has few neurons and synapses, but it implements an ART-1 architecture which is a complicated model involving non-supervised adaptive learning.

Another neural network chip has been announced by INTEL. With the proposed design proposed by INTEL 64 neurons will be implemented in an analog design on each chip. The output of each neuron will be routed to the input of every other neuron on the chip. Variable synaptic weights can be set on the chip, but no on chip learning will be provided. External hardware must be used to train the chip, but this will allow any algorithm including backpropagation to be used.

Another way of implementing neural networks is with digital signal processing integrated circuits which can be incorporated on printed circuit boards to implement complex networks. One example was reported by [24] in which 32 transputers were used in a 4 by 8 array to implement a feedforward ANN with backpropagation. Multiple neurons were sometimes mapped onto a single transputer. Since the network was to be fully connected, the communication problem was a serious concern.

Other experimental analog and digital designs have been simulated or implemented. They usually involve diminishing some aspects of this ideal neural network in order to exploit another aspect needed to solve a particular application. One such example by [25] is a feedforward network in which the synapse weights are restricted to powers of two. In this experiment a simulation model with real values is trained and then the power of 2 weights nearest to the real-value weights is used in the final implementation. Excellent results were obtained when at least nine bits were used in the power of two weights. This method removes the need for multiplications in the simulation process, thereby improving the speed considerably.

8.3 Artificial Neural Network Computers

The majority of hardware that is available at this time is in specialized computers or add-on printed circuit boards. These systems can be either digital or analog. A list of commercially available electronic neural computers compiled by Hect-Nielsen [26] is shown in Table 7.

One of the newer Neuro-Computers, the Delta FPP-2, is primarily an add-on printed circuit card for the IBM-XT/AT and compatible computers. [29] It is a floating point processor that operates at 22 MFLOPS. It uses a Harvard architecture

	DATE	NEURONS	CONNECTS	SPEED C/SEC
ADALINE	1960	1	16	$1.0*10^4$
MADALINE	1962	8	128	$1.0*10^4$
MARK III	1985	$8.0*10^3$	$4.0*10^5$	$3.0*10^5$
ODYSSEY	1986	$8.0*10^3$	$2.5*10^5$	$2.0*10^6$
ANZA	1987	$3.0*10^4$	$5.0*10^5$	$2.5*10^4$
ANZA PLUS	1988	$1.0*10^6$	$1.5*16^6$	$1.5*10^6$
PARALLON 2	1987	$1.0*10^4$	$5.2*10^4$	$1.5*10^4$
PARALLON 2X	1987	$9.1*10^4$	$3.0*10^5$	$1.5*10^4$
DELTA FPP	1987	$1.0*10^6$	$1.0*10^6$	$2.0*10^6$
DELTA FPP-2	1989	$3.1*10^6$	$3.1*10^6$	$2.7*10^6$

Table 7

Commercially Available Neuro Computers

and is heavily pipelined. IEEE 32/64 bit floating point and 32/64 bit integer arithmetic is implemented on this product. Additional software and interface hardware is available to run neural network algorithms and input information from frame grabbers. When simulating an already trained network the speed is increased to 11M connects/second. To demonstrate the importance of using such products the computation time required to train a network using the training and simulation program in Appendix I is shown in Table 8 for various computers.

Computer	Time
AT 286 @ 10Mhz & Coprocessor	1840 Sec
Micro Vax II	700 Sec
AT 386 @ 25MHz & Coprocessor	389 Sec
Cray Y-MP	5 Sec
Delta FPP-2 (estimated from Spec)	.1 Sec (Does not include training signal Generation)

Table 8

Computer Execution Times

These results indicate that when research is being conducted on large neural networks, Floating Point Processors or supercomputers provide a great time savings. This is of particular importance when new networks or new algorithms are being considered. Minor changes in the training rate, momentum, or the method of selecting patterns to be presented to the training algorithm can cause considerable changes in the convergence time or the network tendency to find local instead of global minimum. To find the best parameters to use in the problem considerable testing is required. If fast computing equipment is not used, the time to find good solutions can be prohibitive. These results also demonstrate the importance that realtime analog implementations could play, if they are developed and implemented in commercially available integrated circuits.

8.4 Best ANN Opticns for Feedforward Networks

When considering the best options for the implementation of a project such as this, several additional considerations should be made:

(1) How portable must the system be?

(2) Must the system include local training capability?

(3) How much design effort is required?

(4) What propagation speed are required?

(5) The cost of the system.

If portability, price and speed are not of paramount concern a system such as the DELTA FPP-2 would provide a satisfactory solution. This system can be contained in a portable PC chassis; it can be trained locally, but the cost is over $20,000. This system is already commercially available and operates fast enough for many seismic or sonar applications.

If portability is of importance, a greater design effort will be required to incorporate neural network chips or transputers into an overall design. At the time of this writing network chips that could be used in a feedforward design with backpropagation are just becoming available. Digital implementations using transputers can also provide a suitable solution to most problems, if the price for development is allowable. Since the technology for these transputers is well known and is available with development systems, it could be the preferred method for many systems. However, in the near future analog neural network chips should surpass the digital implementations.

Chapter 9

Conclusions and Recommendations

9.1 Conclusions

The objective of this dissertation has been to demonstrate how narrowband and wideband beamformers can be implemented using artificial neural networks. This objective has been demonstrated through the use of a FORTRAN simulation program and an empirical test which measured seismic data. These tests demonstrate the network's sensitivity to noise, frequency content, amplitude, network topology and sensor topology. A comparison between ANN beamformers and FFT beamformers was also performed.

The empirical data presented in Chapter 6 demonstrates that when the data used to train the network is used to test the same network the success rate is nearly 100%. This success was realized in the presence of both mechanical noise and electrical noise. The network's ability to learn to ignore such noise sources and adhere to the correct signal is one of its strongest attributes. It is however very important to make certain that the training set is representative of all types of signals to be detected. When signals from different distances are used, it is important to train the network using all locations which might effect the spectra of the signal.

In Chapter 6 the comparison between ANN beamformers and FFT beamformers indicated several strengths and limitations for each system. The FFT beamformers are more tolerant of high noise levels than ANN beamformers. FFT beamformers can also be designed to detect arrival angles from multiple sources. However, ANN beamformers are more tolerant of wideband signals and variations in the amplitude of the signal. Both systems gave excellent results when used as narrowband beamformers. The processing effort for both beamformers was similar for simple configurations, but as additional sub-arrays were added to the FFT beamformer to improve its wideband capability the processing effort increased proportionately.

The proper selection of the number of hidden units and the number of input sensors to use are two of the main considerations in the design of a feedforward ANN. Chapter 7 presents and examples of the design criteria for a simulated wideband beamformer. Several plots are also presented to aid future researchers in determining the approximate number of inputs and hidden units required to meet a specific level of acceptance.

9.2 Recommendations

For future study several recommendations are made:

(1) A major problem in training networks with backpropagation

is their convergence to local minima. Many researchers report that by using statistical training methods, such as simulated annealing, the local minima problem can be avoided. If the training times are found to be reasonable, statistical methods could prove to be very useful when actual implementations are made. [27]

(2) There are several additional neural network parameters that should be addressed. One of these parameters is the width of the angle bands. All of the simulations presented in this dissertation used angle bands of 10 degrees. This was chosen to be the maximum size that might be of interest. Using smaller angles will most likely require more sensors and hidden units but a finer resolution could also increase the number of applications for which these networks could be used.

Another important parameter that might be modified in future studies is the number of output neurons in the network. In this study a separate network was used to detect the presence or absence of a signal for each angle. In the early stages of this study it was observed that networks with multiple outputs could be used and trained so different outputs within one network would respond to different angles. These networks require many more hidden units. If the number of hidden units in a system with N outputs is less than N times the number of hidden units in a network with one output, a savings in the number of hidden units could be realized.

(3) Additional tests should be made using actual data and hardware evaluation circuits. An actual implementation can help identify problems that may not be evident from the simulated studies.

(4) Applications should be investigated which involve both the determination of direction and the recognition of a specified target with a fixed spectral pattern. Since the networks were sensitive to changes in the spectra of the transmitting source and were very good at rejecting noise, the combination of these two approaches seems promising.

References

1. Marvin Minsky and Seymore Papert, "Perceptrons an Introduction to Computational Geometry", The MIT Press, 1969.

2. James McClelland and David Rumelhart, "Explorations in Parallel Distributed Processing", The MIT Press, 1988.

3. David Rumelhart and James McClelland, "Parallel Distributed Processing, Volume 1 and Volume 2", The MIT Press, 1986.

4. Philip Wasserman, "Neural Computing, Theory and Practice", Van Nostrand Reinhold, 1989.

5. Randolph Follett and Frank Ingels, "Wideband FFT Beamformer Numerical Simulator", Costal Systems Center Report, Code 4230, 1988.

6. William Knight, Rodger Pridham, and Steven Kay, "Digital Signal Processing for Sonar", Proceedings of the IEEE, Vol. 69, No. 11, November, 1981.

7. Bernard Widrow and Samuel Stearns, "Adaptive Signal Processing", Prentice-Hall, 1985.

8. Carver Mead, "Analog VLSI and Neural Systems", Addison-Wesley, 1989.

9. DARPA Neural Network Study, AFCEA International Press, 1988.

10. R. Paul Gorman and Terrence Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets", Nerual Networks, Vol 1, 1988.

11. Kunihiko Fukushima, Sei Miyake, and Takayukl Ito, "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition", IEEE Transactions of Systems, Man, and Cybernetics 13(5), September/October 1983, pp826-34.

12. Shunji Miyahara, "Automated Radar Target Recognition Based on Models of Neural Nets", PhD. Dissertation, University of Pennsylvania, 1987.

13. Zvi Kohavi, "Switching and Finite Automata Theory", McGraw-Hill, 1978.

14. James Anderson, "Cognitive and Psychological Computation with Neural Models", IEEE Transactions on Systems, Man, and Cybernetics 13(5), September/October, 1983, pp799-815.

15. Michael Cohen and Stephen Grossberg, "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Netwoks", IEEE Transactions on Systems, Man, and Cybernetics, 3(5), September/October 1983, pp815-26.

16. J. H. Justice, N. L. Owsley, J. L. Yen, and A. C. Kak, "Array Signal Processing", Prentice-Hall, 1985.

17. Teuvo Kohonen, "An Introduction to Neural Computing", Neural Networks, Vol 1, 1988, pp3-16.

18. J. J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons", Proceedings of the National Academy of Sciences USA 81, Nay 1984, pp 3088-92.

19. David Tank, John Hopfield, "Simple 'NEURAL' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit", IEEE Transactions on Circuits and Systems 33(5), May 1986, pp533-41.

20. Sung-Kwon Park, "Hopfield Nerual Network for AR Spectral Estimator", IEEE International Symposium on Circuits and Systems, 1990.

21. R. Rastogi, R. K. Gupta, R. Kumaresan, "Array Signal Processing with Interconnected Neuron-like Elements," Proceedings of International Conference on Acoustics, Speech, and Signal Processing, April 1987.

22. Dean R. Collins, P. Andrew Penz, and J. Brock Barton, "Neural Networks Algorithms and Implementations", IEEE International Symposium on Circuits and Systems, 1990, pp2437-40.

23. SYNTONICS Specifications on Dendros-1.

24. I. C. Jou, Y. C. Tsai, "Transputer Based Back-Propagation Neural Net Emulation System", IEEE International Symposium on Circuits and Systems, 1990, pp1879-82.

25. N. Benvenuto, M. Marchesi, F. Piassa, and A. Uncini, "Design of Multi-Layer Neural Networks with Power-of-Two Weights", IEEE International Symposium on Circuits and Systems, 1990, pp 2951-4.

26. Robert Hecht-Neilsen, "Neurocomputing: Picking the Human Brain", IEEE Spectrum 25(3), March 1988, pp36-41.

27. G. K. Atkin, J. E. Bowcock, and N. M. Queen, "Solution of a Distributed Deterministic Parallel Network Using Simulated Annealing", Pattern Recognition, Vol. 22, No. 4, 1989, pp461-466.

28. D. Goryn, M. Kaveh, "Neural Networks for Narrowband and Wideband Direction Finding", Proceedings of International Conference on Acoustics, Speech, and Signal Processing, April 1988.

29. Delta II Plus Specifications, Science Applications International Corporation.

Appendix I

Wideband Beamformer Simulation Program

The following program was written to train and simulate an artificial neural network. The program uses three user supplied files as inputs and provides two sets of files as outputs. The three inputs files are: (1) instruction file, (2) sensor location file, and (3) arrival angle file. The input file contains the names of the sensor file, the arrival angle file, the base name for output files, and all other constants required to train and simulate the network. These constants include; training rate, momentum value, number of training iterations, number of hidden units in each level of the network, the number of frequencies to sum together in training the network, the frequency range of the signal used to train the network, the number of levels in the network, and the amplitude range of the signal that trains the network.

The output files consists of a ".log" file that list the maximum, minimum, and average values of the neuron output when simulated at each arrival angle between 0. And 90. degrees in steps of .5 degrees. The other output files are files that contain the weights of the network and are named ".W01, .W02, ..." to specify which of the arrival angle outputs it represents.

Sample of Input files

Listing of file "WB.INS"

```
LINE10.SEN
FREQ9.BAN
SUN13
0.5 1.0 .8 5 0 0 1 300. 300. 1
30000 0.3 0.9
-30000 0.1 0.9
0 1.0 .9
```

Listing of file "LINE10.SEN"

```
1.0 0.0
2.0 0.0
3.0 0.0
4.0 0.0
5.0 0.0
6.0 0.0
7.0 0.0
8.0 0.0
9.0 0.0
10. 0.0
```

Listing of file "FREQ9.BAN"

```
9
0.0 10.
10. 20.
20. 30.
30. 40.
40. 50.
50. 60.
60. 70.
70. 80.
80. 90.
```

Sample Output Files

```
                     "Instruction information"
FREQ9.BAN
LINE10.SEN
     1    10      5      0      0 60000      1
   300.00000000    300.00000000       0.10000000
0.90000000
      0.00000000     10.00000000       1.00000000
0.50000000
      0.80000000      0.80000000       0.97500000
0.52500000
  77   99    6
  85   99    6
  94   99    6    "Average, maximum and minimum values"
  81   99    6
  98   99   81
  97   99   75
  93   99   77
  77   99    6
  92   99    7
  97   99   76
  98   99   81
  95   99   81
  92   99   70
  96   99   81
  97   99   81
  83   99    7
  99   99   91
  95   99   81
  96   99   81
  80   99    6
  88   99   10
  65   99    6
  49   99    6
  47   92    6
  56   99    6
  42   99    6
  23   90    6
  13   65    6
   6    8    6
   7   18    6
   8   33    6
   6    7    6
   8   46    6
  10   47    6
  13   75    6
   9   64    6
  19   91    6
```

6	9	6
10	87	6
19	82	6
10	72	6
22	81	6
23	89	6
8	28	6
19	95	6
20	80	6
14	91	6
7	17	6
8	34	6
13	75	6
16	99	6
9	58	6
10	85	6
8	42	6
6	7	6
10	81	6
7	11	6
8	43	6
9	57	6
9	30	6
6	6	6
9	57	6
6	6	6
7	19	6
9	40	6
8	40	6
11	95	6
12	98	6
11	76	6
7	16	6
13	74	6
9	58	6
7	8	6
16	93	6
6	9	6
11	65	6
8	27	6
8	34	6
6	6	6
8	41	6
9	38	6
7	12	6
14	90	6
6	6	6
7	10	6
6	7	6
7	10	6

10	78	6
6	6	6
6	8	6
6	7	6
6	6	6
10	86	6
7	21	6
9	42	6
6	6	6
6	6	6
8	21	6
7	11	6
10	72	6
7	19	6
6	6	6
12	82	6
7	11	6
9	48	6
10	48	6
11	76	6
14	90	6
11	87	6
13	78	6
10	70	6
7	11	6
7	20	6
9	57	6
7	12	6
6	9	6
7	10	6
8	39	6
7	13	6
6	6	6
7	28	6
8	25	6
6	6	6
11	39	6
6	7	6
6	6	6
8	33	6
10	76	6
7	23	6
6	7	6
7	9	6
7	23	6
6	8	6
7	19	6
6	7	6
6	6	6
6	6	6

```
      6   8   6
      6   6   6
     10  88   6
      6   6   6
      6   9   6
      6   6   6
      6   6   6
      6   6   6
     11  72   6
      6   7   6
      7  10   6
      9  64   6
     10  71   6
      6   6   6
      6   7   6
      7  15   6
      6   6   6
      7  12   6
      6   7   6
      6   6   6
      6   6   6
      7  26   6
      6   6   6
      6   7   6
      9  54   6
      6   9   6
     10  79   6
      7   9   6
      6   8   6
      6   7   6
     12  80   6
      6   8   6
      6   9   6
      6   7   6
      6   7   6
      8  36   6
      7  14   6
      8  48   6
      6   6   6
      6   7   6
      6   6   6
     10  81   6
      6   8   6
     -1 158  85    "Statiscical information"
     -22998 359    "End of first angle band"
FREQ9.BAN         "Start of next angle band"
LINE10.SEN
         1     10      5      0      0 60000      1
       300.00000000    300.00000000      0.10000000
0.90000000
```

```
      10.00000000       20.00000000       1.00000000
0.50000000
       0.80000000        0.80000000       0.97500000
0.52500000
   5  28   0
   6  52   0
   6  35   0
   7  35   0
   5  35   0
                        "etc."
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                    C
C   Variable Wideband Beamformer Neural Network      C
C                                                    C
C                  VWBBFNN                           C
C                                                    C
C               BY CARY COX                          C
C                                                    C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C             SET NUMBER OF SENSORS AND
C            NUMBER OF ANGELS TO DETECT
C
        PARAMETER (NOS=100, NOF=50)
C
        REAL XIN(NOS),AA(10),F(10)
C
        REAL ANGLO(NOF),ANGHI(NOF)
C
        REAL D(NOS),SX(NOS),SY(NOS)
C
        LOGICAL EXIST,DOSTAT
C
        CHARACTER*4 EXT
        CHARACTER*20 NAMEB
        CHARACTER*20 NAME,NAMES,NAMEF,NAMEO
        CHARACTER*20 NAMEX,NAMESX,NAMEFX
C
        COMMON TRATE,XMO,NIN,NJ,NK,NL,NOUT,OUT,OUTT,XIN
C
C               CONSTANT DATA USED IN PROGRAM
C
        DATA NNN/6715/
        DATA VEL/600./
        DATA PI2/6.283185307/
        DATA DT/.1/
C
        NOUT=1
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C               OPEN MAIN INPUT AND OUTPUT UNITS
C
C
        OPEN(UNIT=11,FILE='WB.INS',FORM='FORMATTED')
C
C               OPEN AND READ ARRAY FILE
C
1111    READ(11,2220,END=1717) NAMES
```

```
2220    FORMAT(A20)
        OPEN(UNIT=10,FILE=NAMES,FORM='FORMATTED')
        REWIND(10)
         INDEX=1
44       READ(10,*,END=1414) SX(INDEX),SY(INDEX)
         INDEX=INDEX+1
         GO TO 44
1414     NIN=INDEX-1
        CLOSE(UNIT=10)
C
C               OPEN  FILTER FILE
C
        READ(11,2220,END=1717) NAMEF
        OPEN(UNIT=10,FILE=NAMEF,FORM='FORMATTED')
        REWIND(10)
        READ(10,*) NFILTZ
         INDEX=1
45       READ(10,*,END=1515) ANGLO(INDEX),ANGHI(INDEX)
         INDEX=INDEX+1
         GO TO 45
1515     NFILT=INDEX-1
        CLOSE(UNIT=10)
C
C           READ NAME-BASE  & NETWORK AND NOISE INSTRUCITONS
C
        READ(11,2220,END=1717) NAMEB
        READ(11,*,END=1717) ATMIN,ATMAX,AN,NJ,NK,NL,
     &       ITYPE,FMIN,FMAX,NFC
        IF((ITYPE.EQ.2).OR.(ITYPE.EQ.3)) NIN=NIN*2
C
C               OPEN "LOG" OUTPUT FILE
C
        NAMEO=NAMEB(1:5)//'.LOG'
        OPEN(UNIT=12,FILE=NAMEO,FORM='FORMATTED',
     &     STATUS='NEW')
C
C               READ NUMBER OF TRAINING ITERATIONS,
C                    TRAINING RATE AND MOMENTUM TERM
C
        NUMTOT=0
2222    READ(11,*,END=1717) NUM1,TRATE,XMO
        IF(NUM1.EQ.0) GO TO 1111
        IF(NUM1.LT.0) THEN
          NUM1=-NUM1
          DOSTAT=.TRUE.
        ELSE
          DOSTAT=.FALSE.
        ENDIF
C
C               CALCULATE TOTAL NUMBER OF ITERATIONS  DONE
```

```
C
      NUMTOT=NUMTOT+NUM1
C
C             TRAIN NETWORK FOR EACH FILTER
C
      DO 9999 NF=1,NFILT
C
C             DESIGN THE WEIGHT FILE NAME
C
      WRITE(EXT,3311) NF
3311  FORMAT('.W',I2.2)
      NAME=NAMEB(1:5)//EXT(1:4)
C
C             CHECK TO SEE IF THIS IS A NEW FILE
C                       OR AN UPDATE
C
      INQUIRE(FILE=NAME,EXIST=EXIST)
C
      IF(.NOT.EXIST) THEN
C
        IF((NK.EQ.0).AND.(NL.EQ.0)) CALL FIXW2
        IF((NK.NE.0).AND.(NL.EQ.0)) CALL FIXW3
        IF((NK.NE.0).AND.(NL.NE.0)) CALL FIXW4
C
C             OPEN THE NEW WEIGHT FILE
C
        OPEN(UNIT=10,FILE=NAME,FORM='FORMATTED',
     &       STATUS='NEW')
C
C             WRITE INSTRUCTION INFO TO WEIGHT FILE
C
        WRITE(10,1010) TRATE,XMO,NIN,NJ,NK,NL,NUMTOT,
     &            ITYPE,NFC,
     &            FMIN,FMAX,
     &            ANGLO(NF),ANGHI(NF),ATMAX,ATMIN,AN,
     &            NAME,NAMEF,NAMES
1010  FORMAT (2F8.4,7I6,7F8.4,3A20)
C
      ELSE
C
C             OPEN OLD FILE AND READ OLD INST
C
        OPEN(UNIT=10,FILE=NAME,FORM='FORMATTED',
     &     STATUS='OLD')
        READ(10,1010) TRATEX,XMOX,NINX,NJX,NKX,NLX,
     &          NUMX,ITYPEX,NFCX,FMINX,FMAXX,
     &          ANGLOX,ANGHIX,ATMAXX,ATMINX,ANX,
     &          NAMEX,NAMEFX,NAMESX
C
C             CK FOR COMPATABILITY
```

```
C
        IF((NINX.NE.NIN).OR.(ANGLOX.NE.ANGLO(NF)).OR.
     &    (ANGHIX.NE.ANGHI(NF)).OR.(ANX.NE.AN).OR.
     &    (ATMINX.NE.ATMIN).OR.(ATMAXX.NE.ATMAX).OR.
     &    (NAMEFX.NE.NAMEF).OR.(NAMESX.NE.NAMES).OR.
     &    (ITYPEX.NE.ITYPE).OR.(NL.NE.NLX).OR.
     &    (NJ.NE.NJX).OR.(NK.NE.NKX).OR.
     &    (FMIN.NE.FMINX).OR.(FMAX.NE.FMAXX)) THEN
        WRITE(*,*) NINX,NIN
        WRITE(*,*) NJX,NJ
        WRITE(*,*) NKX,NK
        WRITE(*,*) NLX,NL
        WRITE(*,*) NFCX,NFC
        WRITE(*,*) FMINX,FMIN
        WRITE(*,*) FMAXX,FMAX
        WRITE(*,*) ANGLOX,ANGLO(NF)
        WRITE(*,*) ANGHIX,ANGHI(NF)
        WRITE(*,*) ANX,AN
        WRITE(*,*) ATMINX,ATMIN
        WRITE(*,*) ATMAXX,ATMAX
         STOP 'ERR #1'
        ENDIF
C
C             READ IN WEIGHTS FOR 2, 3, OR 4 LEVEL ANN
C
        IF((NL.NE.0).AND.(NK.NE.0)) CALL READW4(10)
        IF((NL.EQ.0).AND.(NK.NE.0)) CALL READW3(10)
        IF((NL.EQ.0).AND.(NK.EQ.0)) CALL READW2(10)
C
C             REOPEN THE WEIGHT FILE & REWIND IT
C
        OPEN(UNIT=10,FILE=NAME,FORM='FORMATTED',
     &         STATUS='OLD')
        REWIND(10)
C
C             WRITE THE WEIGHT FILE ID INFORMATION
C
        NUMTOT=NUMX+NUM1
        WRITE(10,1010) TRATE,XMO,NIN,NJ,NK,NL,NUMTOT,
     &             ITYPE,NFC,
     &             FMIN,FMAX,
     &             ANGLO(NF),ANGHI(NF),ATMAX,ATMIN,AN,
     &             NAME,NAMEF,NAMES
      ENDIF
C
C             WRITE INSTRUCTION INFO TO OUTPUT FILE
C
      ANGMAX=ANGHI(NF)*PI2/360.
      ANGMIN=ANGLO(NF)*PI2/360.
```

```
C
      TIME=0.
      WRITE(*,*) 'DO ',NUM1,' ANG =',ANGHI(NF),ANGLO(NF)
C
           DO TRAINING FOR "NUM1" ITERATIONS
C
      DO 987 JJJ=1,NUM1
C
C               SET RANDOM AMPLITUDE WITHIN LIMITS
C                 & SET RANDOM FREQ WITHIN LIMITS
C                 FOR "NC" DIFFERENT FREQUENCIES
C
      DO 877 NC=1,NFC
      AA(NC)=ATMIN+(ATMAX-ATMIN)*RAND(NNN)
      F(NC)=FMIN+(FMAX-FMIN)*RAND(NNN)
877   CONTINUE
C
C         PICK A RANDOM ANGLE BAND TO TRAIN
C
      NUMIT=NFILTZ*2.0+2
      DO 923 IT=1,NUMIT
C
      ITIT=NUMIT*RAND(NNN)+1
      IZZ=NF
      IF(ITIT.LE.NFILTZ) IZZ=ITIT
      IF(ITIT.EQ.NUMIT) IZZ=NF-1
      IF(ITIT.EQ.NUMIT-1) IZZ=NF+1
      IF(IZZ.GT.NFILTZ) IZZ=1
      IF(IZZ.LE.0) IZZ=NFILTZ
C
C               SET A RANDOM ANGLE WITHIN THE BAND
C
      RANGE=ANGHI(IZZ)-ANGLO(IZZ)
      RANGE=RANGE*PI2/360.
      ANGR=(FLOAT(IZZ-1)+RAND(NNN))*RANGE
C
C               SET THE TARGET VALUES
C
      OUTT=0.0
      IF((ANGR.GT.ANGMIN).AND.(ANGR.LT.ANGMAX)) OUTT=1.0
C
C               CALCULATE X & Y COORDINATES OF THIS ANGLE
C                         (RANGE=1000 FEET)
C
      X=1000.*SIN(ANGR)
      Y=1000.*COS(ANGR)
C
C               GET THE SELECTED TYPE OF INPUTS
C
C               DO A LINEAR  SELETCION IF ITYPE=1
```

```
C
        IF(ITYPE.EQ.1) THEN
C
         CALL WAVSET(D,SX,SY,NIN,X,Y,VEL)
C
         DO 11 IS=1,NIN
         XIN(IS)=WAVE(IS,D,AA,F,NFC,AN,TIME)
11       CONTINUE
        ENDIF
C
C              DO AN ELIPTICAL SELECTION IF ITYPE=2
C
        IF(ITYPE.EQ.2) THEN
C
         NIN2=NIN/2
         CALL WAVSET(D,SX,SY,NIN2,X,Y,VEL)
C
         DO 117 IS=1,NIN2
         XIN(IS)=WAVE(IS,D,AA,F,NFC,AN,TIME)
         XIN(IS+NIN2)=XIN(IS)*XIN(IS)
117      CONTINUE
        ENDIF
C
C              DO A X=SIN(A); Y=SIN(A+P); ETC.
C                 SELECTION IF ITYPE=3
C
        IF(ITYPE.EQ.3) THEN
C
         NIN2=NIN/2
         CALL WAVSET(D,SX,SY,NIN2,X,Y,VEL)
C
         DO 116 IS=1,NIN2
         XIN(IS)=WAVE(IS,D,AA,F,NFC,AN,TIME)
         XIN(IS+NIN2)=XIN(IS)*XIN(IS)
         IF(IS.EQ.1) THEN
          X1=XIN(1)
         ELSE
          XIN(IS-1)=XIN(IS-1)*XIN(IS)
         ENDIF
116      CONTINUE
         XIN(NIN2)=XIN(NIN2)*X1
        ENDIF
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                        C
C              TRAIN THE NETWORK         C
C                                        C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
```

```
      IF((NK.EQ.0).AND.(NL.EQ.0)) THEN
C
              CALL ANN2
              CALL LEARN2
              CALL ADJUST2
      ENDIF
C
      IF((NK.NE.0).AND.(NL.EQ.0)) THEN
C
              CALL ANN3
              CALL LEARN3
              CALL ADJUST3
C
      ENDIF
C
      IF((NK.NE.0).AND.(NL.NE.0)) THEN
C
              CALL ANN4
              CALL LEARN4
              CALL ADJUST4
C
      ENDIF
C                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
923   CONTINUE
C
C             PICK A RANDOM TIME FOR TEXT ITERATION
C
      TIME=TIME+DT*RAND(NNN)
C
987   CONTINUE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                     C
C             TRAINING IS COMPLETE    C
C                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C             SAVE WEIGHTS
C
      IF((NK.EQ.0).AND.(NL.EQ.0)) CALL SAVEW2(10)
      IF((NK.NE.0).AND.(NL.EQ.0)) CALL SAVEW3(10)
      IF((NK.NE.0).AND.(NL.NE.0)) CALL SAVEW4(10)
C
9999  CONTINUE
C
C             DO STATISTICS IF REQUESTED
C
      IF(DOSTAT) THEN
```

```
        CALL STATS(NFILT,NAMEB,D,SX,SY,NFC)
        ENDIF
C
C               GO DO ANOTHER TEST
C
        GO TO 2222
C
C               ALL TEST ARE DONE -- EXIT
C
1717    CLOSE(UNIT=11)
        CLOSE(UNIT=12)
        STOP 'DONE'
        END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C               SUBROUTINE TO SIMULATE THE TRAINED
C               NETWORK AND COMPILE SOME STATISTICS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE STATS(NFF,NAMEB,D,SX,SY,NFC)
C
        PARAMETER (NOS=100, NOF=50)
C
C
        REAL XIN(NOS)
C
        REAL ANGLO(NOF),ANGHI(NOF)
C
        REAL D(NOS),SX(NOS),SY(NOS),AA(10),F(10)
C
        CHARACTER*4 EXT
        CHARACTER*20 NAMEB
        CHARACTER*20 NAME,NAMES,NAMEF
C
        COMMON TRATE,XMO,NIN,NJ,NK,NL,NOUT,OUT,OUTT,XIN
C
C               CONSTANT DATA USED IN PROGRAM
C
        DATA NNN/6715/
        DATA VEL/600./
        DATA PI2/6.283185307/
        DATA DT/.1/
C
        NOUT=1
C
C               LOOK FOR ALL VERSIONS OF THE BASE FILE
C
        DO 9999 NF=1,NFF
```

```
C
C               DESIGN THE WEIGHT FILE NAME
C
      WRITE(EXT,3311) NF
3311  FORMAT('.W',I2.2)
      NAME=NAMEB(1:5)//EXT(1:4)
C
C               OPEN OLD FILE AND READ OLD INST
C
      WRITE(*,2211) NAME
2211  FORMAT (' TRY TO OPEN ',A20)
      OPEN(UNIT=10,FILE=NAME,FORM='FORMATTED',
     &     STATUS='OLD')
      REWIND(10)
      READ(10,1010) TRATE,XMO,NIN,NJ,NK,NL,NUM,ITYPE,NFC,
     &              FMIN,FMAX,
     &              ANGLO(NF),ANGHI(NF),ATMAX,ATMIN,AN,
     &              NAME,NAMEF,NAMES
1010  FORMAT (2F8.4,7I6,7F8.4,3A20)
C
C               READ WEIGHTS FOR 2, 3, OR 4 LEVEL ANN
C
      IF((NK.EQ.0).AND.(NL.EQ.0)) CALL READW2(10)
      IF((NK.NE.0).AND.(NL.EQ.0)) CALL READW3(10)
      IF((NK.NE.0).AND.(NL.NE.0)) CALL READW4(10)
C
C               GGET ANGLE (IN RAD)
C
      ANGMAX=ANGHI(NF)*PI2/360.
      ANGMIN=ANGLO(NF)*PI2/360.
C
      TIME=0.
C
C               SET AMPLITUDE RANGE TO
C              90% OF TRAINED AMPLITUDE
C
      RANGE=(.1*(ATMAX-ATMIN))/2.
      SATMAX=ATMAX-RANGE
      SATMIN=ATMIN+RANGE
      SAN=AN
C
C               WRITE IMPORTRAN VARIABLES TO "LOG" FILE
C
      WRITE(12,1212) NAMEF
      WRITE(12,1212) NAMES
      WRITE(12,1213) NFC,NIN,NJ,NK,NL,NUM,ITYPE
      WRITE(12,1214) FMIN,FMAX,TRATE,XMO
      WRITE(12,1214) ANGLO(NF),ANGHI(NF),ATMAX,ATMIN
      WRITE(12,1214) AN,SAN,SATMAX,SATMIN
1212  FORMAT(A20)
```

```
1213    FORMAT (7I6)
1214    FORMAT (4F16.8)
C
        IERRT=0
        IXLOT=0
        IXHIT=0
        IAMBT=0
C
C                 TEST THE NETWORK FROM 1 TO 90 DEGREES IN
C                             STEPS OF .5 DEGREES
C
        DO 987 JJJ=1,180
C
        ANGR=FLOAT(JJJ)*PI2/(2.*360.)
C
C                    SET RANDOM AMPLITUDE AND
C                 RANDOM FREQUENCY WITHIN LIMITS
C
        DO 571 NC=1,NFC
        AA(NC)=SATMIN+(SATMAX-SATMIN)*RAND(NNN)
        F(NC)=FMIN+(FMAX-FMIN)*RAND(NNN)
571     CONTINUE
C
C           SET THE TARGET VALUE
C
        AVG=0.
        XMAX=0.
        XMIN=1.
        IXLO=0
        IXHI=0
        IAMBIG=0
C
C                 FOR EACH ANGLE TEST THE NETWORK
C                      AT 20 RANDOM TIMES
C
        DO 923 IT=1,20
C
C                 SET TARGET VALUES
C
        OUTT=0.0
        IF((ANGR.GT.ANGMIN).AND.(ANGR.LT.ANGMAX)) OUTT=1.0
C
C                 CALCULATE X & Y COORDINATES OF THIS ANGLE
C
        X=1000.*SIN(ANGR)
        Y=1000.*COS(ANGR)
C
C                 GET THE SELECTED TYPE OF INPUTS
C
C                 DO A LINEAR  SELETCION IF ITYPE=1
```

```
C
        IF(ITYPE.EQ.1) THEN
C
         CALL WAVSET(D,SX,SY,NIN,X,Y,VEL)
C
         DO 11 IS=1,NIN
         XIN(IS)=WAVE(IS,D,AA,F,NFC,AN,TIME)
11       CONTINUE
        ENDIF
C
C               DO AN ELIPTICAL SELECTION IF ITYPE=2
C
        IF(ITYPE.EQ.2) THEN
C
         NIN2=NIN/2
         CALL WAVSET(D,SX,SY,NIN2,X,Y,VEL)
C
         DO 117 IS=1,NIN2
         XIN(IS)=WAVE(IS,D,AA,F,NFC,AN,TIME)
         XIN(IS+NIN2)=XIN(IS)*XIN(IS)
117      CONTINUE
        ENDIF
C
C               DO A X=SIN(A); Y=SIN(A+P); ETC.
C                    SELECTION IF ITYPE=3
C
        IF(ITYPE.EQ.3) THEN
C
         NIN2=NIN/2
         CALL WAVSET(D,SX,SY,NIN2,X,Y,VEL)
C
         DO 116 IS=1,NIN2
         XIN(IS)=WAVE(IS,D,AA,F,NFC,AN,TIME)
         XIN(IS+NIN2)=XIN(IS)*XIN(IS)
         IF(IS.EQ.1) THEN
          X1=XIN(1)
         ELSE
          XIN(IS-1)=XIN(IS-1)*XIN(IS)
         ENDIF
116      CONTINUE
         XIN(NIN2)=XIN(NIN2)*X1
        ENDIF
C
C               DO SIMULATION FOR 2, 3, OR 4 LEVEL ANN
C
        IF((NK.EQ.0).AND.(NL.EQ.0)) CALL ANN2
        IF((NK.NE.0).AND.(NL.EQ.0)) CALL ANN3
        IF((NK.NE.0).AND.(NL.NE.0)) CALL ANN4
C
C               FIND AVG, MAX, & MIN OUTPUT
```

```
C
        AVG=AVG+OUT
        IF(OUT.GT.XMAX) XMAX=OUT
        IF(OUT.LT.XMIN) XMIN=OUT
C
C               FIND STATISTICS ON GOOD, BAD,
C                  & AMBIGUOUS RESULTS
C
        I ''(OUTT.LT.0.2).AND.(OUT.LT.0.2)) IXLO=IXLO+1
        I  (OUTT.GT.0.7).AND.(OUT.GT.0.7)) IXHI=IXHI+1
        IF((OUT.GT.0.2).AND.(OUT.LT.0.7)) IAMBIG=IAMBIG+1
        IERROR=20-IXLO-IXHI-IAMBIG
C
C               SET RANDOM TIME FOR NEXT ITERATION
C
        TIME=TIME+DT*RAND(NNN)
923     CONTINUE
C
C               COMPUTE AVERAGE
C
        AVG=AVG/20.
C
C               WRITE RESULTS TO "LOG" FILE SCALED BY 100
C
        IAVG=AVG*100.
        MAX=XMAX*100.
        MIN=XMIN*100.
        WRITE(12,1233) IAVG,MAX,MIN
1233    FORMAT (3I4)
C
C               COMPUTE STATISTICS
C
        IERRT=IERRT+IERROR
        IXLOT=IXLOT+IXLO
        IXHIT=IXHIT+IXHI
        IAMBT=IAMBT+IAMBIG
C
987     CONTINUE
C
        WRITE(12,1233) -1,IERRT,IAMBT
        WRITE(12,1233) -2,IXLOT,IXHIT
C
C
C
9999    CONTINUE
C
C               ALL ANGLE BANDS ARE FINISHED
C
```

```
      RETURN
C
      END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     SUBROUTINE TO SET UP THE PROPOGATION
C        TIME FOR ALL "NIN" SENSORS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      SUBROUTINE WAVSET(D,SX,SY,NS,X,Y,VEL)
      REAL D(NS),SX(NS),SY(NS)
C
      DO 1 I=1,NS
C
C             CALCUALTE DISTANCES FORM SENSORS
C
      XX=SX(I)-X
      YY=SY(I)-Y
C
C             CALCULATE TIME DELAY TO SENSOR
C
      D(I) = XX*XX + YY*YY
      D(I) = SQRT(D(I))/VEL
1     CONTINUE
      RETURN
      END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     CALCULATE THE VALUE AT SENSOR "IS" AT TIME "TIME"
C             AND FREQUENCY "F" AND AMPLITUDE "A"
C                  AND NOISE LEVEL "AN"
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      FUNCTION WAVE(IS,D,A,F,NC,AN,TIME)
      REAL D(1),A(NC),F(NC)
      DATA PI2,N/6.283185307,6713/
C
C             GET TOTAL TIME
C     (PROPAGATION TIME + REAL TIME)
C
      T = TIME + D(IS)
C
C             CALCULATE WAVE AMPLITUDE AT THIS SENSOR
C             (NOISE + ALL FREQUENCY COMPONENTS)
C
      WAVE= AN*(RAND(N)-.5)
```

```
        DO 33 I=1,NC
        W=F(I)*PI2
        WAVE = A(I)*SIN(W*T) + WAVE
33      CONTINUE
        RETURN
        END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C               FIRST DERIVATIVE OF PERCEPTRON'S
C                     NONLINEAR FUNCTION
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        FUNCTION FUNP(XX)
        FUNP=XX*(1.-XX)
        RETURN
        END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C               PERCEPTRON'S NONLINEAR FUNCTION
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        FUNCTION FUN(X)
        FUN=1./(1.+EXP(-X))
        RETURN
        END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       RANDOM NUMBER GEN.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        FUNCTION RAND(N)
        K=31627
        N=N*K
        N=MOD(N,32768)
        RN=N
        RN=RN/32767.
        RAND=ABS(RN)
        RETURN
        END
```

```
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C                FOR 4 LEVEL ANN
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C                SET RANDOM INITIAL WEIGHTS
C
        SUBROUTINE FIXW4
C
        PARAMETER (NOS=100,NOF=50)
C
        REAL WIJ(NOS,NOS),WJK(NOS,NOS)
        REAL WKL(NOS,NOS),WLO(NOS,1)
        REAL BIASJ(NOS),BIASL(NOS),BIASK(NOS),BIASO(1)
        REAL OUTJ(NOS),OUTK(NOS),OUTL(NOS),OUT(1)
        REAL EL(NOS),EK(NOS),EJ(NOS),EO(1)
        REAL DKL(NOS,NOS),DJK(NOS,NOS)
        REAL DIJ(NOS,NOS),DLO(NOS,1)
        REAL DBIASL(NOS),DBIASK(NOS),DBIASJ(NOS),DBIASO(1)
        REAL OUTT(1),XIN(NOS)
C
        COMMON/STUFF/WIJ,WJK,WKL,WLO,BIASJ,BIASL,
     &   BIASK,BIASO,OUTL,OUTJ,OUTK,EL,EK,EJ,EO,DKL,
     &   DJK,DIJ,DLO,DBIASL,DBIASK,DBIASJ,DBIASO
C
        COMMON TRATE,XMO,NIN,NJ,NK,NL,NOUT,OUT,OUTT,XIN
C
        DATA DDD/50./
        DATA NNN/6713/
C
        DO 444 II=1,NIN
        DO 44 JJ=1,NJ
        WIJ(II,JJ)=-.5+RAND(NNN)
        WIJ(II,JJ)=WIJ(II,JJ)/DDD
44      CONTINUE
444     CONTINUE
C
        DO 445 JJ=1,NJ
        BIASJ(JJ)=-.5+RAND(NNN)
        BIASJ(JJ)=BIASJ(JJ)/DDD
        DO 45 KK=1,NK
        WJK(JJ,KK)=-.5+RAND(NNN)
        WJK(JJ,KK)=WJK(JJ,KK)/DDD
45      CONTINUE
445     CONTINUE
C
        DO 446 KK=1,NK
        BIASK(KK)=-.5+RAND(NNN)
```

```
          BIASK(KK)=BIASK(KK)/DDD
          DO 46 LL=1,NL
          WKL(KK,LL)=-.5+RAND(NNN)
          WKL(KK,LL)=WKL(KK,LL)/DDD
46        CONTINUE
446       CONTINUE
C
          DO 546 LL=1,NL
          BIASL(LL)=-.5+RAND(NNN)
          BIASL(LL)=BIASL(LL)/DDD
          DO 54 MM=1,NOUT
          WLO(LL,MM)=-.5+RAND(NNN)
          WLO(LL,MM)=WLO(LL,MM)/DDD
54        CONTINUE
546       CONTINUE
C
          DO 47 MM=1,NOUT
          BIASO(MM)=-.5+RAND(NNN)
          BIASO(MM)=BIASO(MM)/DDD
47        CONTINUE
          RETURN
C
C                 SAVE WEIGHTS
C
          ENTRY SAVEW4(IUNIT)
          WRITE(IUNIT,1010) ((WIJ(II,JJ),II=1,NIN),JJ=1,NJ),
     &                  (BIASJ(JJ),JJ=1,NJ)
          WRITE(IUNIT,1010) ((WJK(JJ,KK),JJ=1,NJ),KK=1,NK),
     &                  (BIASK(KK),KK=1,NK)
          WRITE(IUNIT,1010) ((WKL(KK,LL),KK=1,NK),LL=1,NL),
     &                  (BIASL(LL),LL=1,NL)
          WRITE(IUNIT,1010) ((WLO(LL,MM),LL=1,NL),MM=1,NOUT),
     &                  (BIASO(MM),MM=1,NOUT)
1010      FORMAT (F16.8)
          CLOSE(UNIT=IUNIT)
          RETURN
C
C                 READ WEIGHTS
C
          ENTRY READW4(IUNIT)
C
          READ(IUNIT,1010) ((WIJ(II,JJ),II=1,NIN),JJ=1,NJ),
     &                  (BIASJ(JJ),JJ=1,NJ)
          READ(IUNIT,1010) ((WJK(JJ,KK),JJ=1,NJ),KK=1,NK),
     &                  (BIASK(KK),KK=1,NK)
          READ(IUNIT,1010) ((WKL(KK,LL),KK=1,NK),LL=1,NL),
     &                  (BIASL(LL),LL=1,NL)
          READ(IUNIT,1010) ((WLO(LL,MM),LL=1,NL),MM=1,NOUT),
     &                  (BIASO(MM),MM=1,NOUT)
          CLOSE(UNIT=IUNIT)
```

```
      RETURN
C
C                    ROUTINE TO SIMULATE ANN
C
      ENTRY ANN4
C
C                SIMULATE LEVEL 1
C
      DO 10 J=1,NJ
      S=BIASJ(J)
      DO 11 I=1,NIN
      S=S+XIN(I)*WIJ(I,J)
11    CONTINUE
      OUTJ(J)=FUN(S)
10    CONTINUE
C
C                SIMULATE LEVEL 2
C
      DO 20 K=1,NK
      S=BIASK(K)
      DO 21 J=1,NJ
      S=S+OUTJ(J)*WJK(J,K)
21    CONTINUE
      OUTK(K)=FUN(S)
20    CONTINUE
C
C                SIMULATE LEVEL 3
C
      DO 30 L=1,NL
      S=BIASL(L)
      DO 31 K=1,NK
      S=S+OUTK(K)*WKL(K,L)
31    CONTINUE
      OUTL(L)=FUN(S)
30    CONTINUE
C
C                SIMULATE LEVEL 4
C
      DO 40 M=1,NOUT
      S=BIASO(M)
      DO 41 L=1,NL
      S=S+OUTL(L)*WLO(L,M)
41    CONTINUE
      OUT(M)=FUN(S)
40    CONTINUE
C
      RETURN
C
C                LEARNING ROUTINE
C
```

```
      ENTRY LEARN4
C
C              TRAIN LEVEL 4
C
      DO 110 M=1,NOUT
      EL(M)=FUNP(OUT(M))*(OUTT(M)-OUT(M))
      DBIASO(M)=TRATE*EO(M)+XMO*DBIASO(M)
      DO 110 L=1,NL
      DLO(L,M)=TRATE*EO(M)*OUTL(L)+XMO*DLO(L,M)
110    CONTINUE
C
C             TRAIN LEVEL 3
C
      DO 115 L=1,NL
      S=0.
      DO 116 M=1,NOUT
      S=S+WLO(L,M)*EO(M)
116   CONTINUE
      EL(L)=FUNP(OUTL(L))*S
      DBIASL(L)=TRATE*EL(L)+XMO*DBIASL(L)
      DO 117 K=1,NK
      DKL(K,L)=TRATE*EL(L)*OUTK(K)+XMO*DKL(K,L)
117   CONTINUE
115   CONTINUE
C
C             TRAIN LEVEL 2
C
      DO 120 K=1,NK
      S=0.
      DO 121 L=1,NL
      S=S+WKL(K,L)*EL(L)
121    CONTINUE
      EK(K)=FUNP(OUTK(K))*S
      DBIASK(K)=TRATE*EK(K)+XMO*DBIASK(K)
      DO 120 J=1,NJ
      DJK(J,K)=TRATE*EK(K)*OUTJ(J)+XMO*DJK(J,K)
120    CONTINUE
C
C             TRAIN LEVEL 1
C
      DO 130 J=1,NJ
      S=0.
      DO 131 K=1,NK
      S=S+WJK(J,K)*EK(K)
131    CONTINUE
      EJ(J)=FUNP(OUTJ(J))*S
      DBIASJ(J)=TRATE*EJ(J)+XMO*DBIASJ(J)
      DO 130 I=1,NIN
      DIJ(I,J)=TRATE*EJ(J)*XIN(I)+XMO*DIJ(I,J)
130    CONTINUE
```

```
C
        RETURN
C
C               ROUTINE TO ADJUST WEIGHTS
C
        ENTRY ADJUST4
C
C               ADJUST BIAS WEIGHTS
C
        DO 229 M=1,NOUT
229     BIASO(M)=BIASO(M)+DBIASO(M)
C
        DO 230 L=1,NL
230      BIASL(L)=BIASL(L)+DBIASL(L)
C
        DO 220 K=1,NK
220      BIASK(K)=BIASK(K)+DBIASK(K)
C
        DO 210 J=1,NJ
210      BIASJ(J)=BIASJ(J)+DBIASJ(J)
C
C               ADJUST WEIGHTS
C
        DO 239 M=1,NOUT
        DO 239 L=1,NL
239     WLO(L,M)=WLO(L,M)+DLO(L,M)
C
        DO 240 L=1,NL
        DO 240 K=1,NK
240      WKL(K,L)=WKL(K,L)+DKL(K,L)
C
        DO 250 K=1,NK
        DO 250 J=1,NJ
250      WJK(J,K)=WJK(J,K)+DJK(J,K)
C
        DO 260 J=1,NJ
        DO 260 I=1,NIN
260      WIJ(I,J)=WIJ(I,J)+DIJ(I,J)

C
        RETURN
        END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C               3 LEVEL ANN
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C               SET RANDOM INITIAL WEIGHTS
```

```
C
      SUBROUTINE FIXW3
C
      PARAMETER (NOS=100,NOF=50)
C
      REAL WIJ(NOS,NOS),WJK(NOS,NOS),WKL(NOS,1)
      REAL BIASJ(NOS),BIASL(1),BIASK(NOS)
      REAL OUTJ(NOS),OUTK(NOS),OUT(1)
      REAL EL(1),EK(NOS),EJ(NOS)
      REAL DKL(NOS,1),DJK(NOS,NOS),DIJ(NOS,NOS)
      REAL DBIASL(1),DBIASK(NOS),DBIASJ(NOS)
      REAL OUTT(1),XIN(NOS)
C
      COMMON/STUFF/WIJ,WJK,WKL,BIASJ,BIASL,BIASK,
     &  OUTJ,OUTK,EL,EK,EJ,DKL,DJK,DIJ,
     &  DBIASL,DBIASK,DBIASJ
C
      COMMON TRATE,XMO,NIN,NJ,NK,NL,NOUT,OUT,OUTT,XIN
C
      DATA DDD/50./
      DATA NNN/6713/
C
      DO 444 II=1,NIN
      DO 44 JJ=1,NJ
      WIJ(II,JJ)=-.5+RAND(NNN)
      WIJ(II,JJ)=WIJ(II,JJ)/DDD
44    CONTINUE
444   CONTINUE
C
      DO 445 JJ=1,NJ
      BIASJ(JJ)=-.5+RAND(NNN)
      BIASJ(JJ)=BIASJ(JJ)/DDD
      DO 45 KK=1,NK
      WJK(JJ,KK)=-.5+RAND(NNN)
      WJK(JJ,KK)=WJK(JJ,KK)/DDD
45    CONTINUE
445   CONTINUE
C
      DO 446 KK=1,NK
      BIASK(KK)=-.5+RAND(NNN)
      BIASK(KK)=BIASK(KK)/DDD
      DO 46 LL=1,NOUT
      WKL(KK,LL)=-.5+RAND(NNN)
      WKL(KK,LL)=WKL(KK,LL)/DDD
46    CONTINUE
446   CONTINUE
C
      DO 47 LL=1,NOUT
      BIASL(LL)=-.5+RAND(NNN)
      BIASL(LL)=BIASL(LL)/DDD
```

```
47      CONTINUE
        RETURN
C
C                SAVE WEIGHTS
C
        ENTRY SAVEW3(IUNIT)
        WRITE(IUNIT,1010) ((WIJ(II,JJ),II=1,NIN),JJ=1,NJ),
     &                 (BIASJ(JJ),JJ=1,NJ)
        WRITE(IUNIT,1010) ((WJK(JJ,KK),JJ=1,NJ),KK=1,NK),
     &                 (BIASK(KK),KK=1,NK)
        WRITE(IUNIT,1010) ((WKL(KK,LL),KK=1,NK),LL=1,NOUT),
     &                 (BIASL(LL),LL=1,NOUT)
1010    FORMAT (F16.8)
        CLOSE(UNIT=IUNIT)
        RETURN
C
C                READ WEIGHTS
C
        ENTRY READW3(IUNIT)
C
        READ(IUNIT,1010) ((WIJ(II,JJ),II=1,NIN),JJ=1,NJ),
     &                 (BIASJ(JJ),JJ=1,NJ)
        READ(IUNIT,1010) ((WJK(JJ,KK),JJ=1,NJ),KK=1,NK),
     &                 (BIASK(KK),KK=1,NK)
        READ(IUNIT,1010) ((WKL(KK,LL),KK=1,NK),LL=1,NOUT),
     &                 (BIASL(LL),LL=1,NOUT)
        CLOSE(UNIT=IUNIT)
        RETURN
C
C                  ROUTINE TO SIMULATE ANN
C
        ENTRY ANN3
C
C                SIMULATE LEVEL 1
C
        DO 10 J=1,NJ
        S=BIASJ(J)
        DO 11 I=1,NIN
        S=S+XIN(I)*WIJ(I,J)
11      CONTINUE
        OUTJ(J)=FUN(S)
10      CONTINUE
C
C                SIMULATE LEVEL 2
C
        DO 20 K=1,NK
        S=BIASK(K)
        DO 21 J=1,NJ
        S=S+OUTJ(J)*WJK(J,K)
21      CONTINUE
```

```
        OUTK(K)=FUN(S)
20      CONTINUE
C
C               SIMULATE LEVEL 3
C
        DO 30 L=1,NOUT
        S=BIASL(L)
        DO 31 K=1,NK
        S=S+OUTK(K)*WKL(K,L)
31      CONTINUE
        OUT(L)=FUN(S)
30      CONTINUE
C
        RETURN
C
C               LEARNING ROUTINE
C
        ENTRY LEARN3
C
C               TRAIN LEVEL 3
C
        DO 110 L=1,NOUT
        EL(L)=FUNP(OUT(L))*(OUTT(L)-OUT(L))
        DBIASL(L)=TRATE*EL(L)+XMO*DBIASL(L)
        DO 110 K=1,NK
        DKL(K,L)=TRATE*EL(L)*OUTK(K)+XMO*DKL(K,L)
110      CONTINUE
C
C               TRAIN LEVEL 2
C
        DO 120 K=1,NK
        S=0.
        DO 121 L=1,NOUT
        S=S+WKL(K,L)*EL(L)
121      CONTINUE
        EK(K)=FUNP(OUTK(K))*S
        DBIASK(K)=TRATE*EK(K)+XMO*DBIASK(K)
        DO 120 J=1,NJ
        DJK(J,K)=TRATE*EK(K)*OUTJ(J)+XMO*DJK(J,K)
120      CONTINUE
C
C               TRAIN LEVEL 1
C
        DO 130 J=1,NJ
        S=0.
        DO 131 K=1,NK
        S=S+WJK(J,K)*EK(K)
131      CONTINUE
        EJ(J)=FUNP(OUTJ(J))*S
        DBIASJ(J)=TRATE*EJ(J)+XMO*DBIASJ(J)
        DO 130 I=1,NIN
        DIJ(I,J)=TRATE*EJ(J)*XIN(I)+XMO*DIJ(I,J)
130      CONTINUE
C
        RETURN
C
C               ROUTINE TO ADJUST WEIGHTS
C
        ENTRY ADJUST3
C
C               ADJUST BIAS WEIGHTS
C
        DO 230 L=1,NOUT
230      BIASL(L)=BIASL(L)+DBIASL(L)
C
        DO 220 K=1,NK
220      BIASK(K)=BIASK(K)+DBIASK(K)
C
        DO 210 J=1,NJ
210      BIASJ(J)=BIASJ(J)+DBIASJ(J)
C
C               ADJUST WEIGHTS
C
        DO 240 L=1,NOUT
        DO 240 K=1,NK
240      WKL(K,L)=WKL(K,L)+DKL(K,L)
C
        DO 250 K=1,NK
        DO 250 J=1,NJ
250      WJK(J,K)=WJK(J,K)+DJK(J,K)
C
        DO 260 J=1,NJ
        DO 260 I=1,NIN
260      WIJ(I,J)=WIJ(I,J)+DIJ(I,J)
C
        RETURN
        END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C               2 LEVEL ANN
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C               SET RANDOM INITIAL WEIGHTS
C
        SUBROUTINE FIXW2
C
        PARAMETER (NOS=100,NOF=50)
C
```

```
      REAL WIJ(NOS,NOS),WJL(NOS,1)
      REAL BIASJ(NOS),BIASL(1)
      REAL OUTJ(NOS)
      REAL EL(1),EJ(NOS)
      REAL DJL(NOS,1),DIJ(NOS,NOS)
      REAL DBIASL(1),DBIASJ(NOS)
      REAL OUTT(1),XIN(NOS),OUT(1)
C
      COMMON/STUFF/WIJ,WJL,BIASJ,BIASL,
     &  OUTJ,EL,EJ,DJL,DIJ,
     &  DBIASL,DBIASJ
C
      COMMON TRATE,XMO,NIN,NJ,NK,NL,NOUT,OUT,OUTT,XIN
C
      DATA NNN/6713/
      DATA DDD/50./
C
      DO 444 II=1,NIN
      DO 44 JJ=1,NJ
      WIJ(II,JJ)=-.5+RAND(NNN)
      WIJ(II,JJ)=WIJ(II,JJ)/DDD
44    CONTINUE
444   CONTINUE
C
      DO 446 JJ=1,NJ
      BIASJ(JJ)=-.5+RAND(NNN)
      BIASJ(JJ)=BIASJ(JJ)/DDD
      DO 46 LL=1,NOUT
      WJL(JJ,LL)=-.5+RAND(NNN)
      WJL(JJ,LL)=WJL(JJ,LL)/DDD
46    CONTINUE
446   CONTINUE
C
      DO 47 LL=1,NOUT
      BIASL(LL)=-.5+RAND(NNN)
      BIASL(LL)=BIASL(LL)/DDD
47    CONTINUE
      RETURN
C
C               SAVE WEIGHTS
C
      ENTRY SAVEW2(IUNIT)
      WRITE(IUNIT,1010) ((WIJ(II,JJ),II=1,NIN),JJ=1,NJ),
     &              (BIASJ(JJ),JJ=1,NJ)
      WRITE(IUNIT,1010) ((WJL(JJ,LL),JJ=1,NJ),LL=1,NOUT),
     &              (BIASL(LL),LL=1,NOUT)
1010  FORMAT (F16.8)
      CLOSE(UNIT=IUNIT)
      RETURN
C
```

```
C                 READ WEIGHTS
C
        ENTRY READW2(IUNIT)
C
        READ(IUNIT,1010) ((WIJ(II,JJ),II=1,NIN),JJ=1,NJ),
     &              (BIASJ(JJ),JJ=1,NJ)
        READ(IUNIT,1010) ((WJL(JJ,LL),JJ=1,NJ),LL=1,NOUT),
     &              (BIASL(LL),LL=1,NOUT)
        CLOSE(UNIT=IUNIT)
        RETURN
C
C                   ROUTINE TO SIMULATE ANN
C
        ENTRY ANN2
C
C                 SIMULATE LEVEL 1
C
        DO 10 J=1,NJ
        S=BIASJ(J)
        DO 11 I=1,NIN
        S=S+XIN(I)*WIJ(I,J)
11      CONTINUE
        OUTJ(J)=FUN(S)
10      CONTINUE
C
C                 SIMULATE LEVEL 2
C
        DO 30 L=1,NOUT
        S=BIASL(L)
        DO 31 J=1,NJ
        S=S+OUTJ(J)*WJL(J,L)
31      CONTINUE
        OUT(L)=FUN(S)
30      CONTINUE
C
        RETURN
C
C                 LEARNING ROUTINE
C
        ENTRY LEARN2
C
C                 TRAIN LEVEL 2
C
        DO 110 L=1,NOUT
        EL(L)=FUNP(OUT(L))*(OUTT(L)-OUT(L))
        DBIASL(L)=TRATE*EL(L)+XMO*DBIASL(L)
        DO 110 J=1,NJ
        DJL(J,L)=TRATE*EL(L)*OUTJ(J)+XMO*DJL(J,L)
110      CONTINUE
C
```

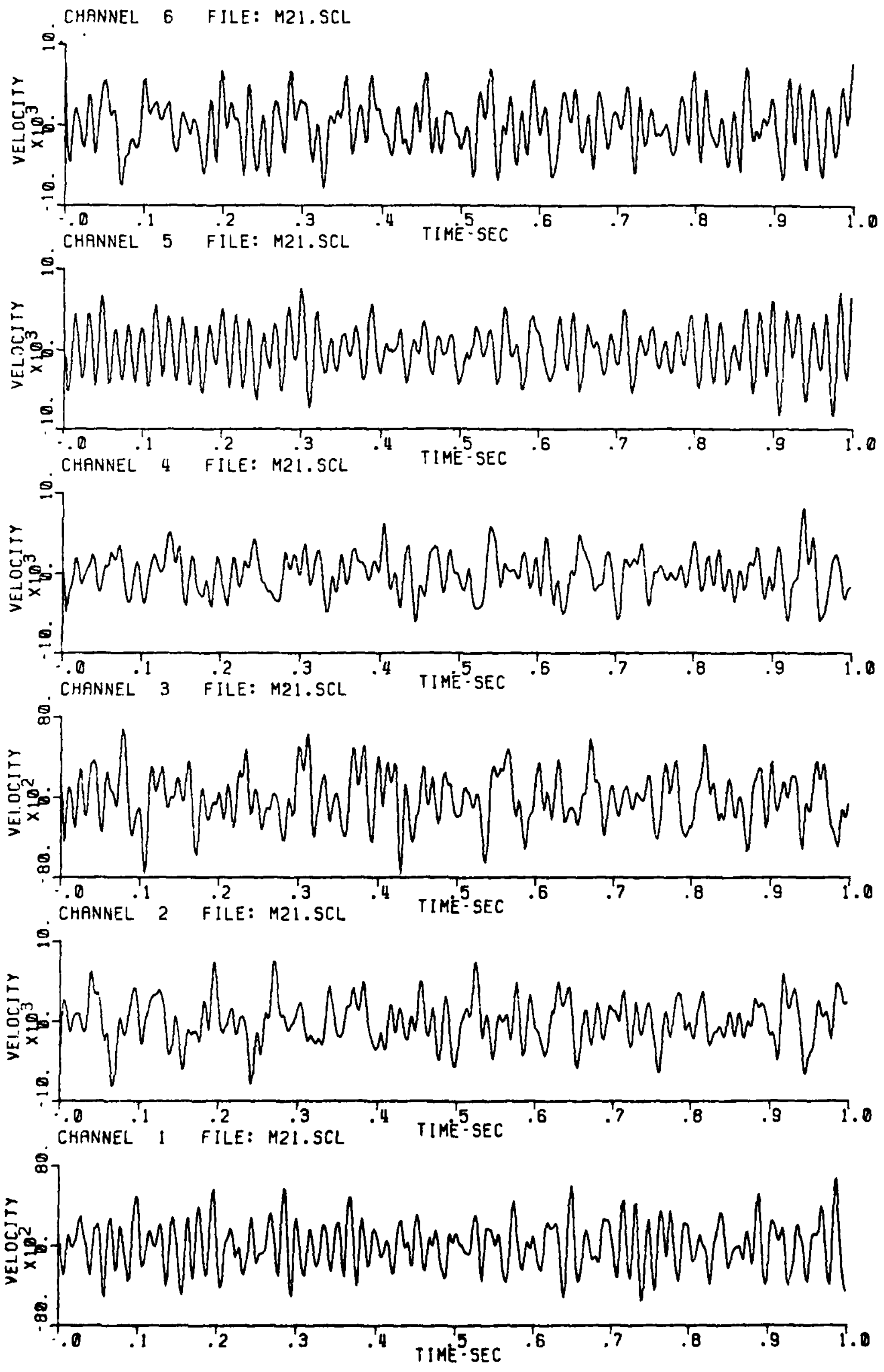
```
C                TRAIN LEVEL 1
C
        DO 130 J=1,NJ
        S=0.
        DO 131 L=1,NOUT
        S=S+WJL(J,L)*EL(L)
131      CONTINUE
        EJ(J)=FUNP(OUTJ(J))*S
        DBIASJ(J)=TRATE*EJ(J)+XMO*DBIASJ(J)
        DO 130 I=1,NIN
        DIJ(I,J)=TRATE*EJ(J)*XIN(I)+XMO*DIJ(I,J)
130      CONTINUE
C
        RETURN
C
C                ROUTINE TO ADJUST WEIGHTS
C
        ENTRY ADJUST2
C
C                ADJUST BIAS WEIGHTS
C
        DO 230 L=1,NOUT
230      BIASL(L)=BIASL(L)+DBIASL(L)
C
        DO 210 J=1,NJ
210      BIASJ(J)=BIASJ(J)+DBIASJ(J)
C
C                ADJUST WEIGHTS
C
        DO 240 L=1,NOUT
        DO 240 J=1,NJ
240      WJL(J,L)=WJL(J,L)+DJL(J,L)
C
        DO 260 J=1,NJ
        DO 260 I=1,NIN
260      WIJ(I,J)=WIJ(I,J)+DIJ(I,J)
C
        RETURN
        END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C                END OF VWBBFNN
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

Appendix II

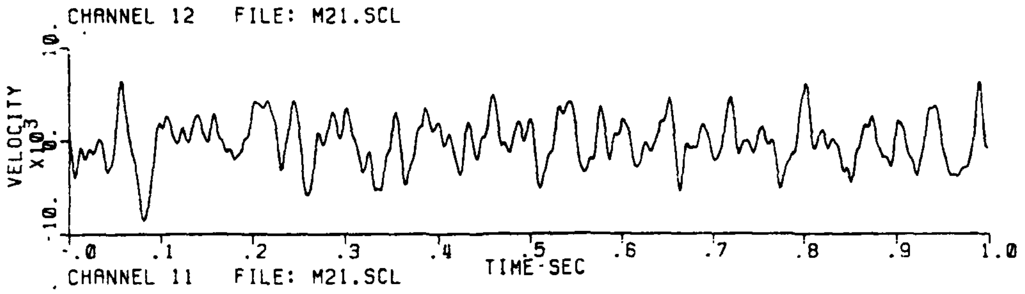
Time History Plots of Seismic Data

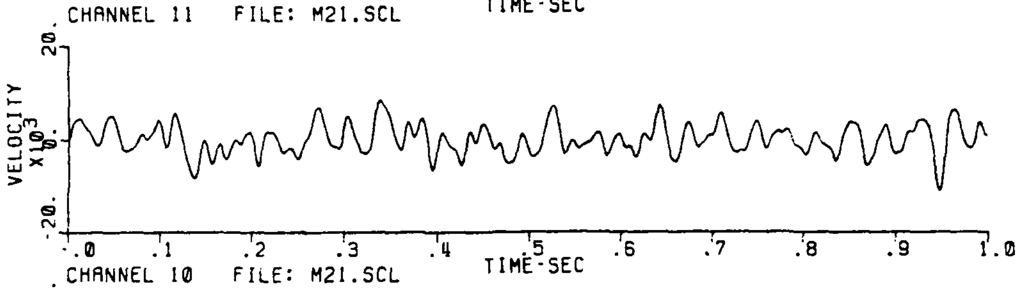
The following data was acquired on analog magnetic tape and digitized for processing. Neither amplitude or time scaling was required to train the network.

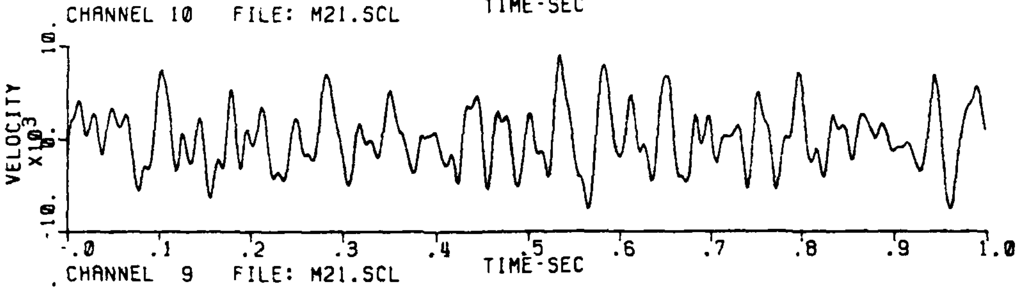
Channels 1-6 Seismic Data Direction A

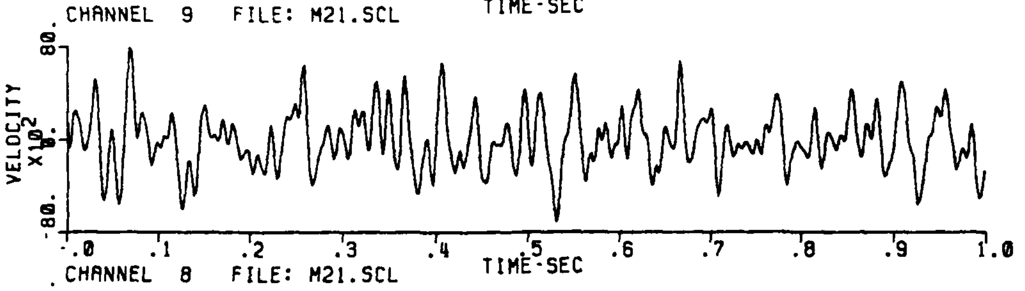


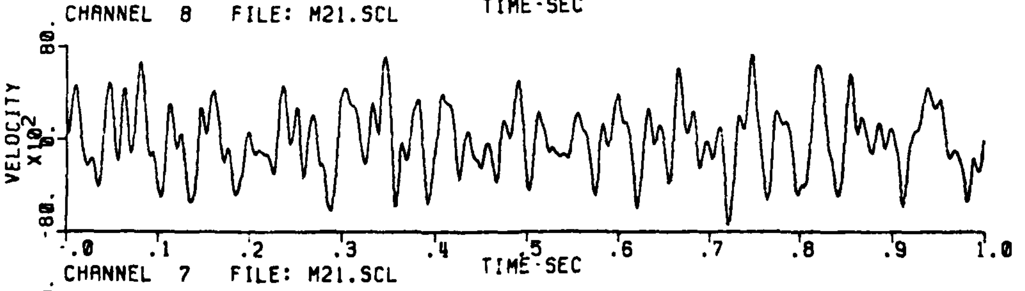
Channels 7-12 Seismic Data Direction A

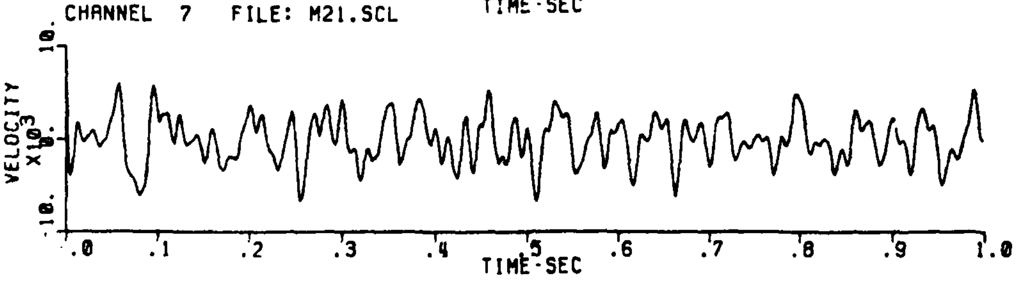




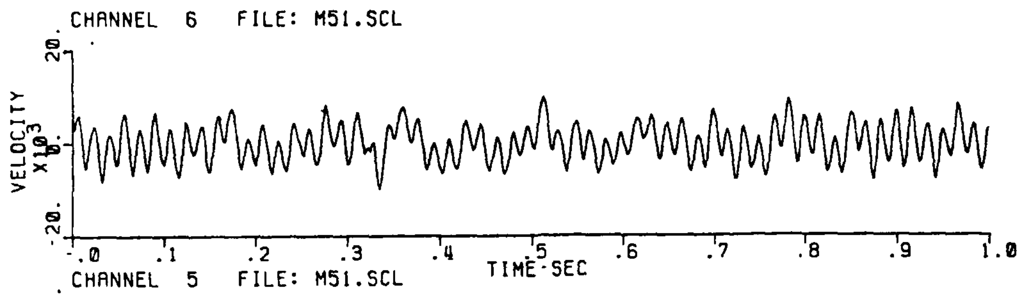


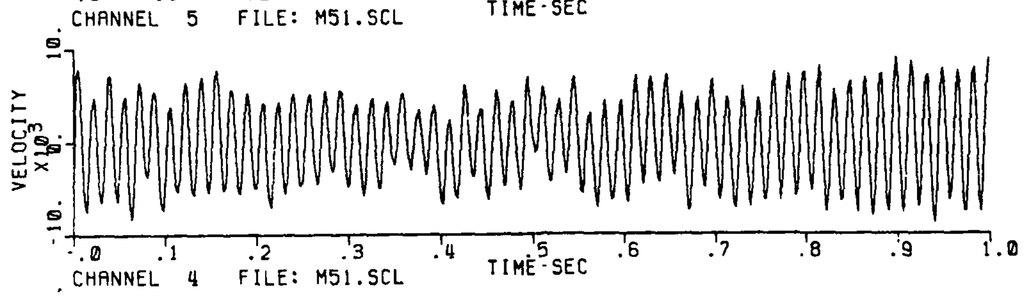


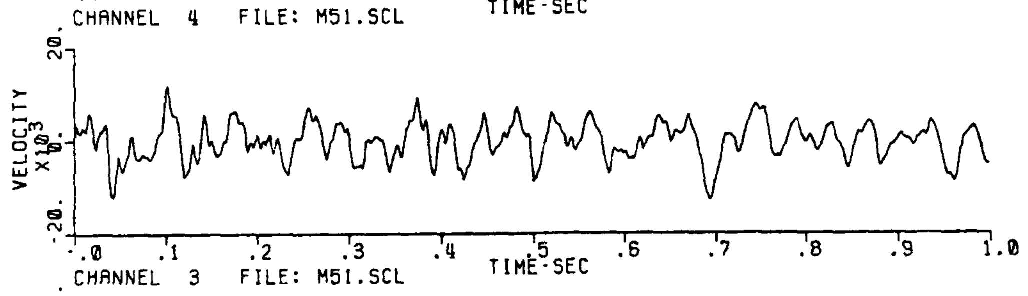


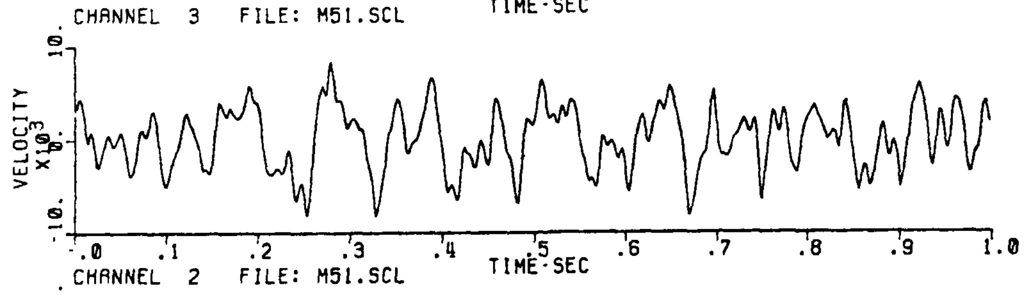


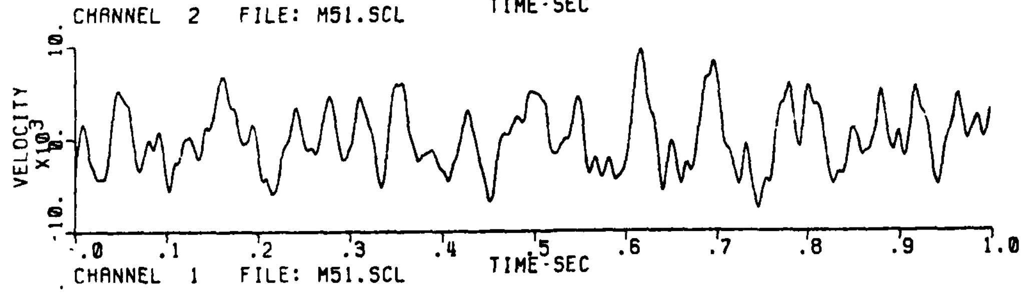
Channels 1-6 Seismic Data Direction B

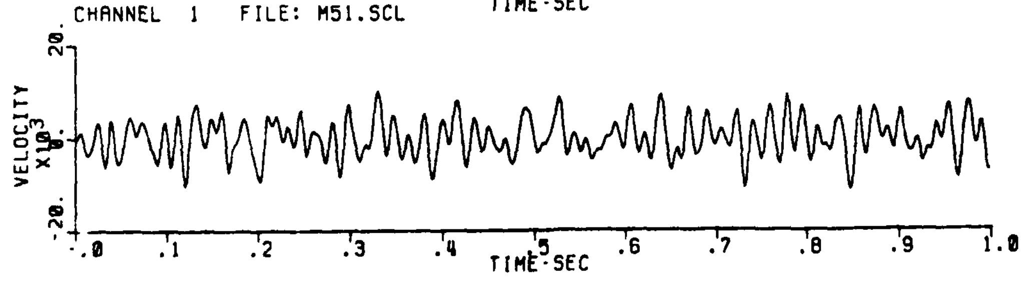




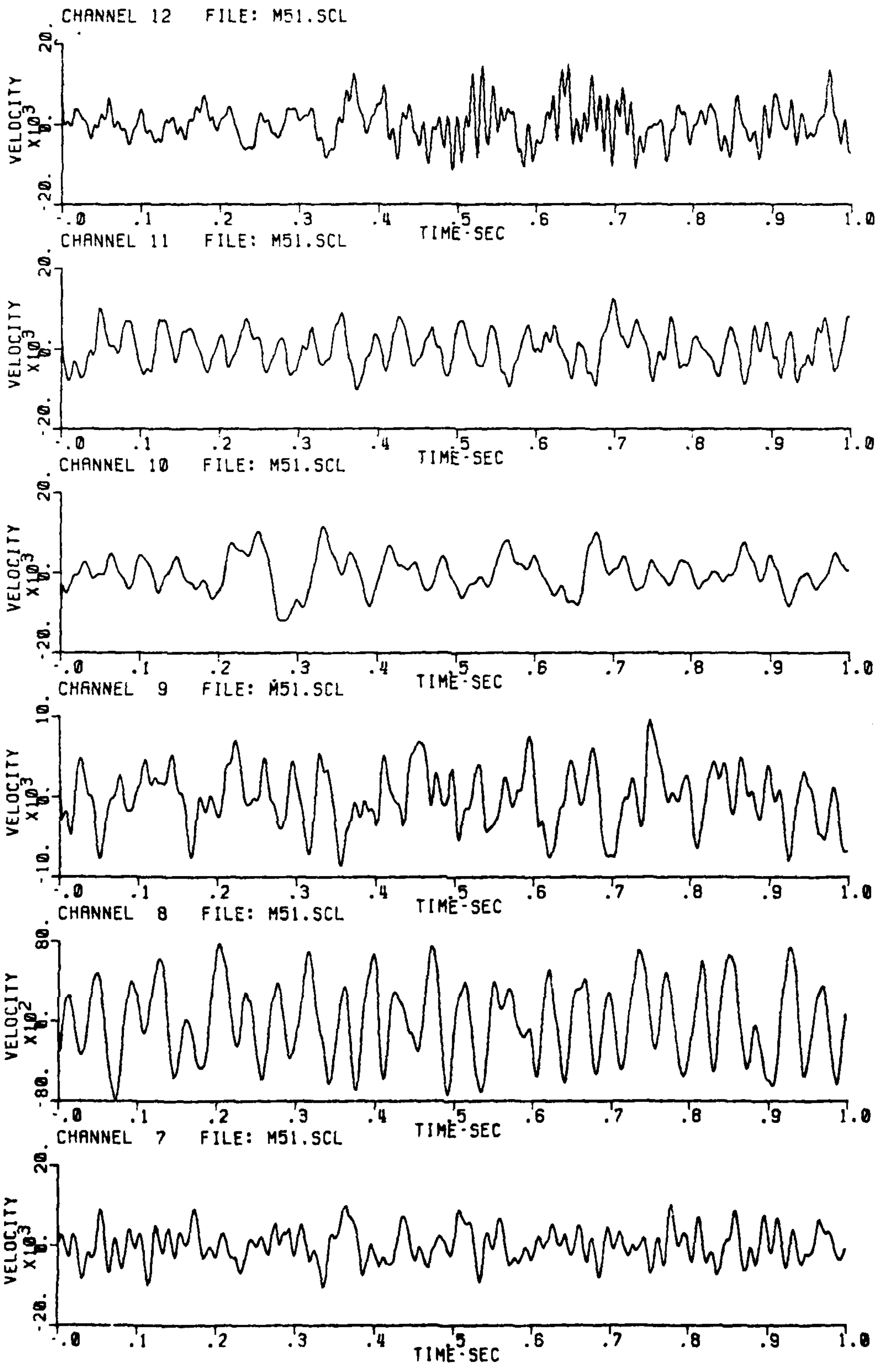




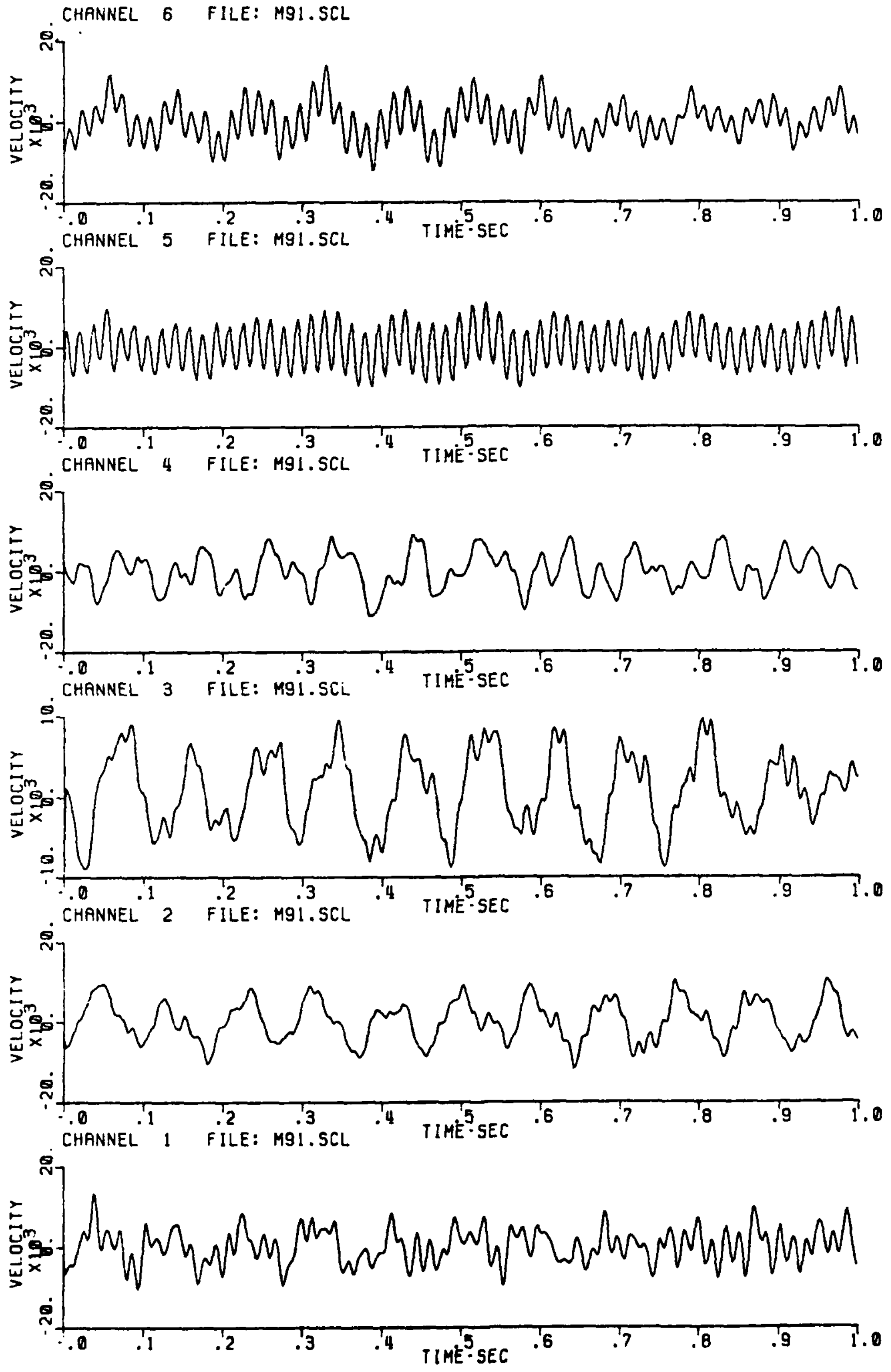




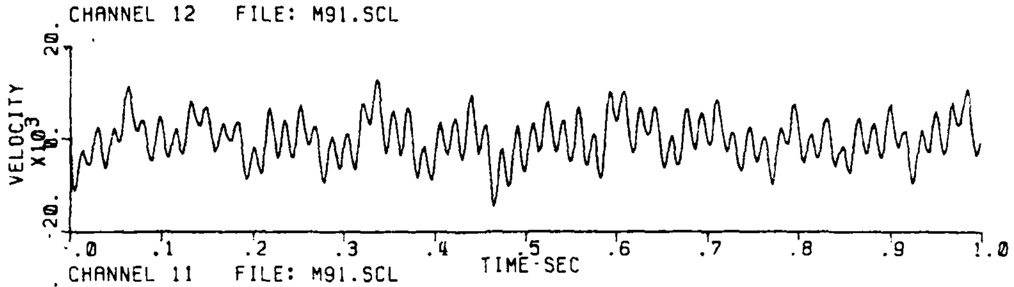
Channels 7-12 Seismic Data Direction B

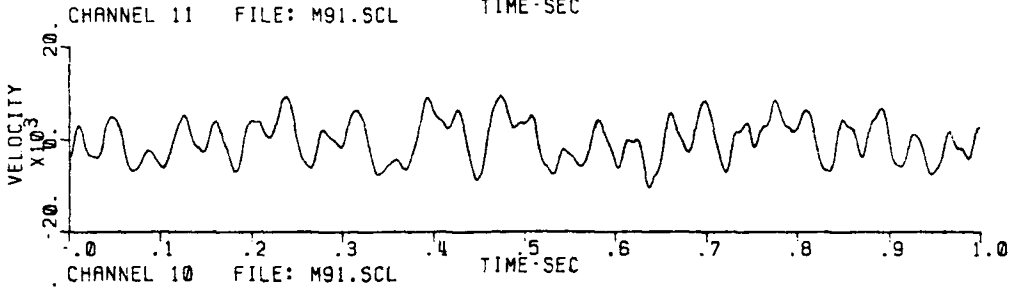


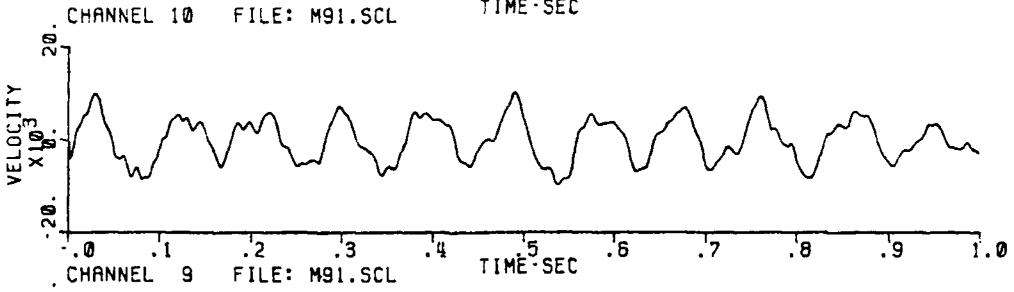
Channels 1-6 Seismic Data Direction C

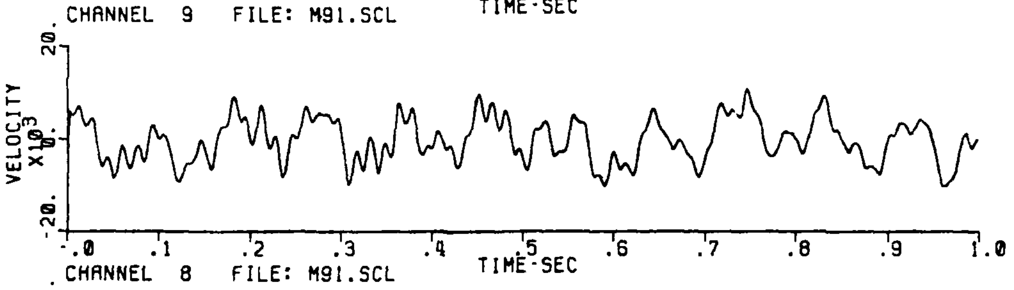


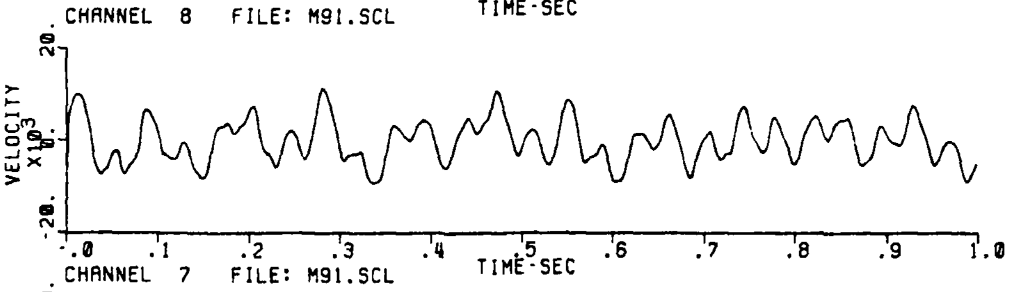
Channels 7-12 Seismic Data Direction C

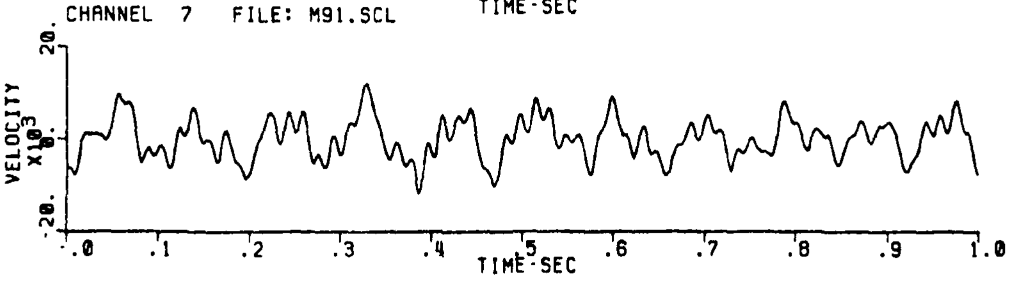




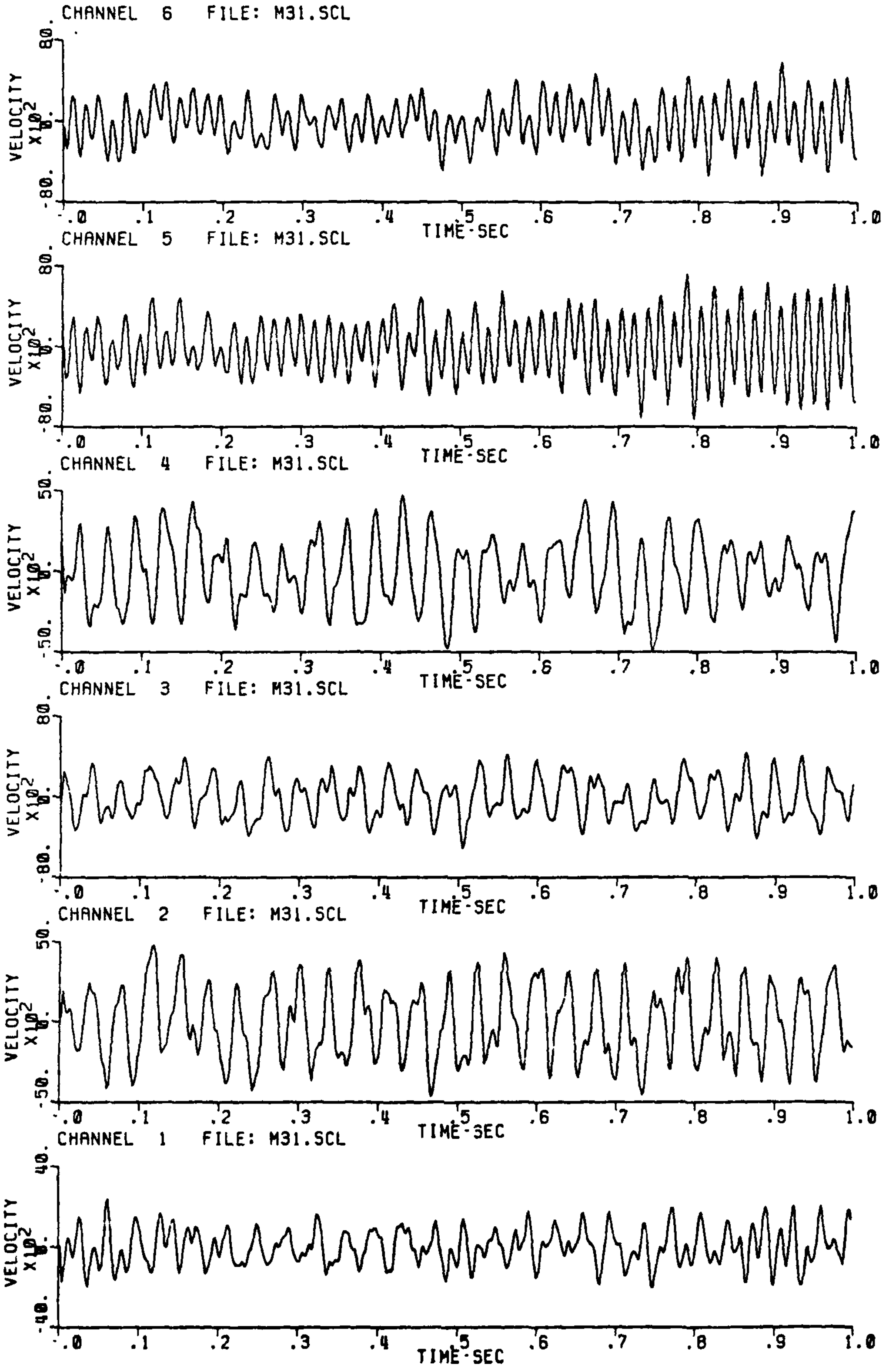




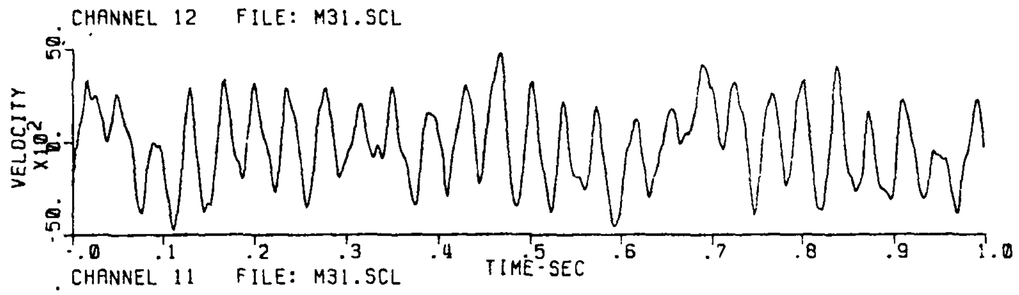


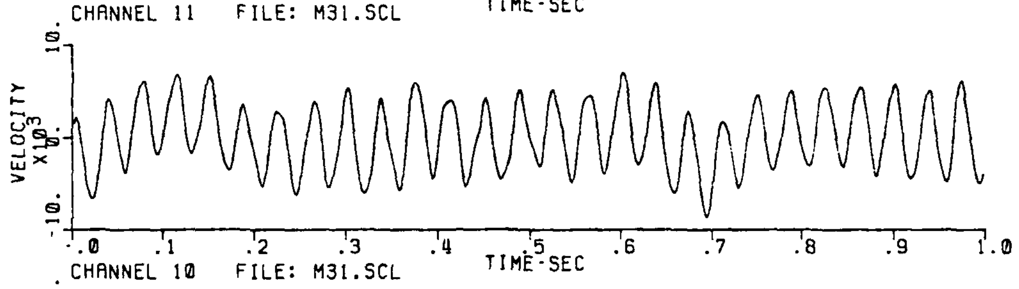


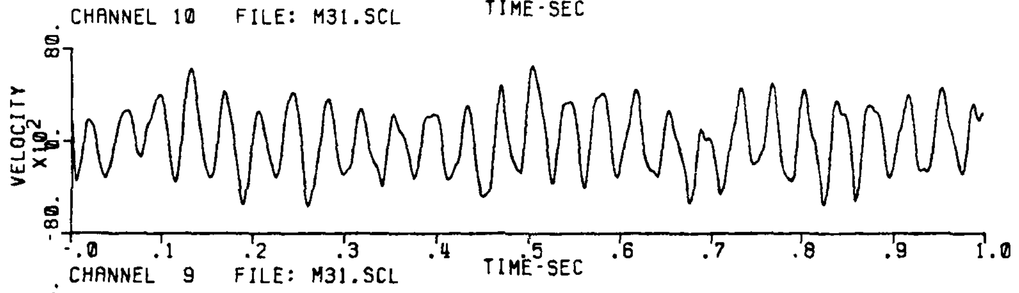
Channels 1-6 Seismic Data Direction A Range 2

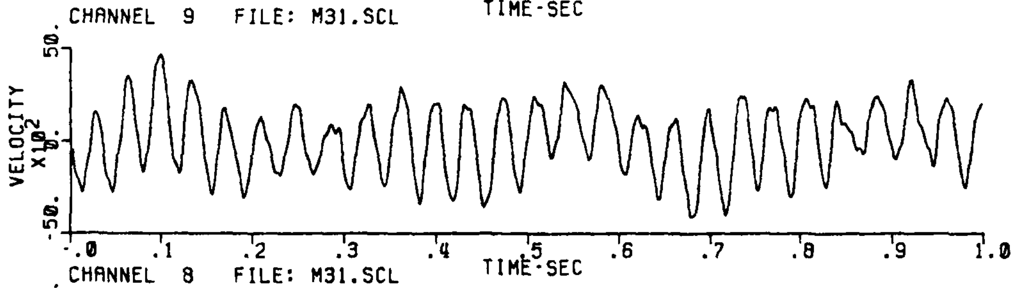


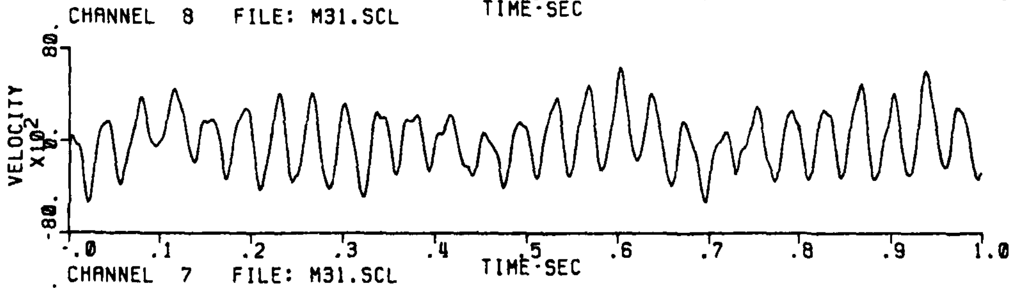
Channels 7-12 Seismic Data Direction A Range 2

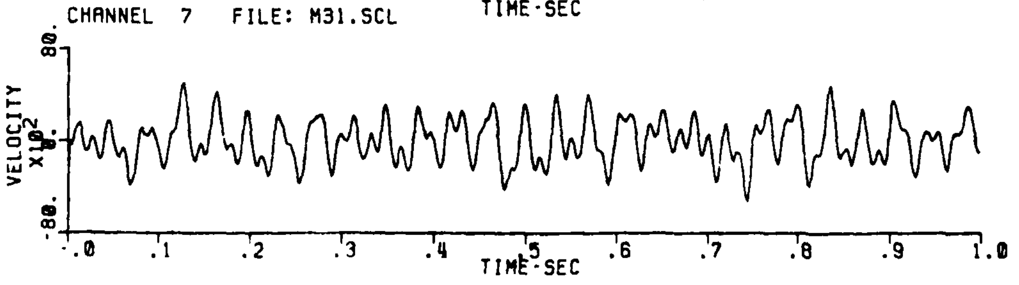






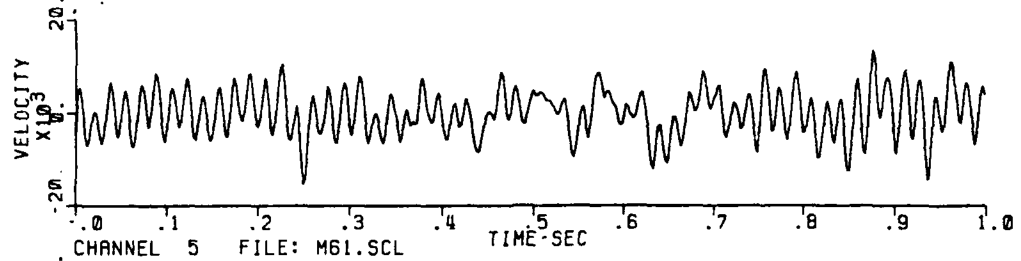




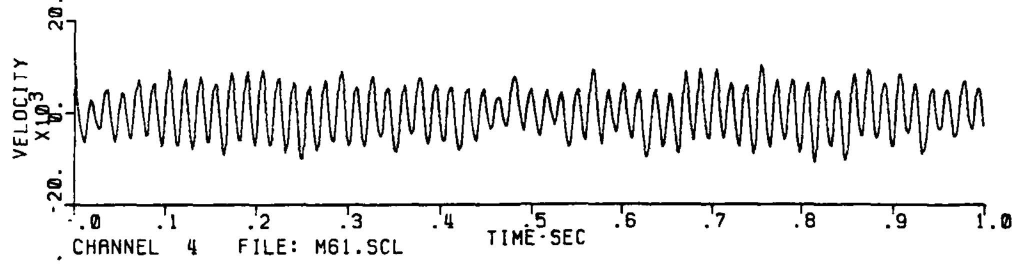


Channels 1-6 Seismic Data Direction B Range 2

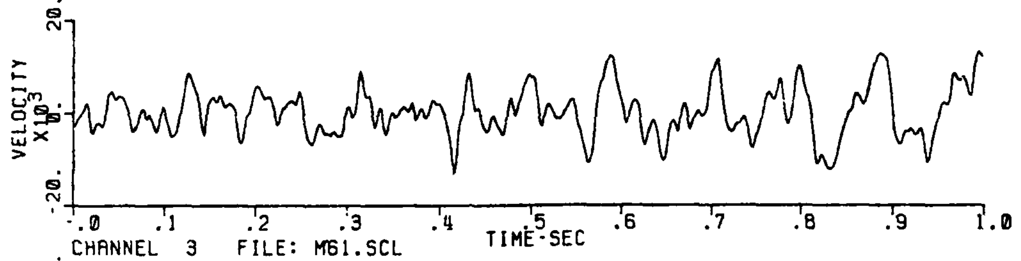
CHANNEL 6 FILE: M61.SCL



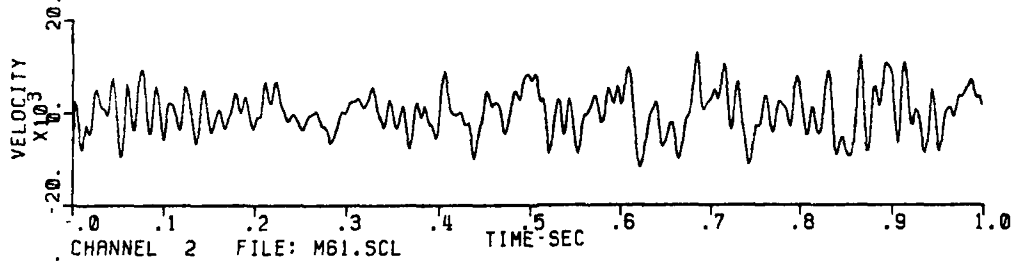
CHANNEL 5 FILE: M61.SCL



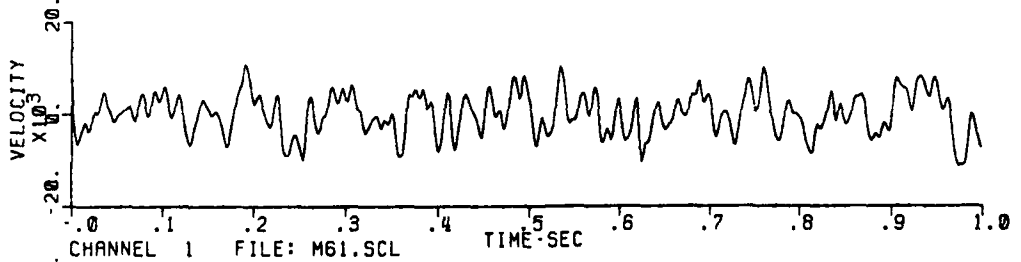
CHANNEL 4 FILE: M61.SCL



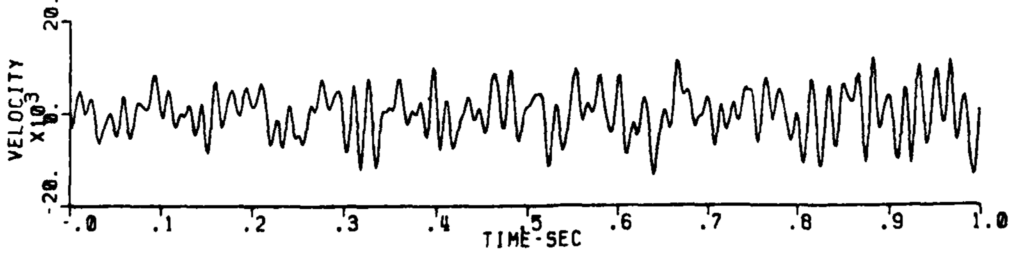
CHANNEL 3 FILE: M61.SCL



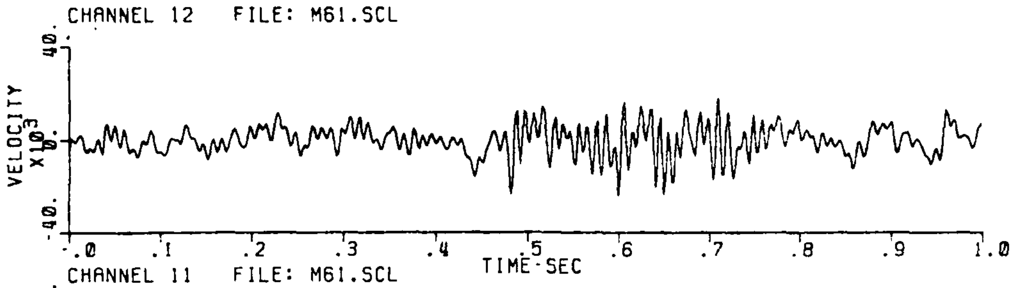
CHANNEL 2 FILE: M61.SCL

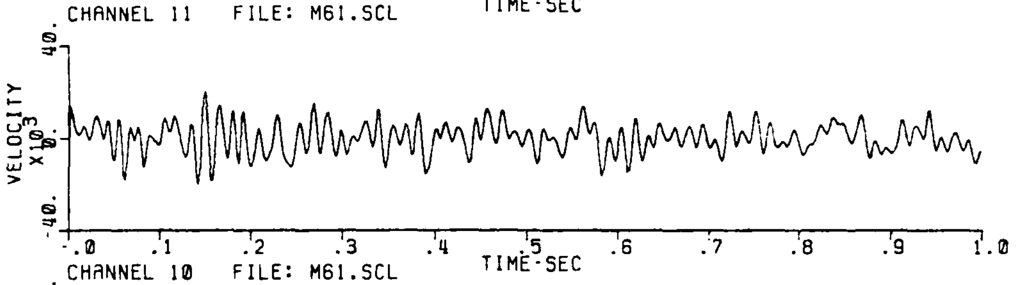


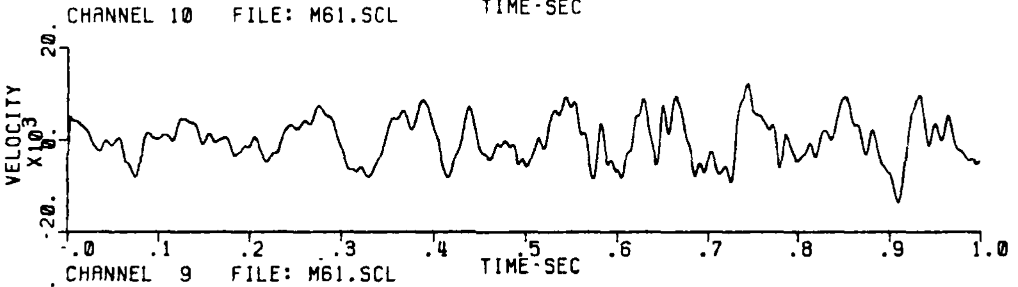
CHANNEL 1 FILE: M61.SCL

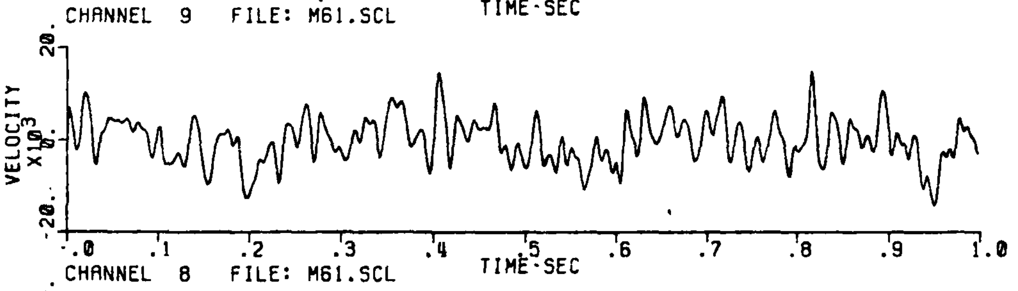


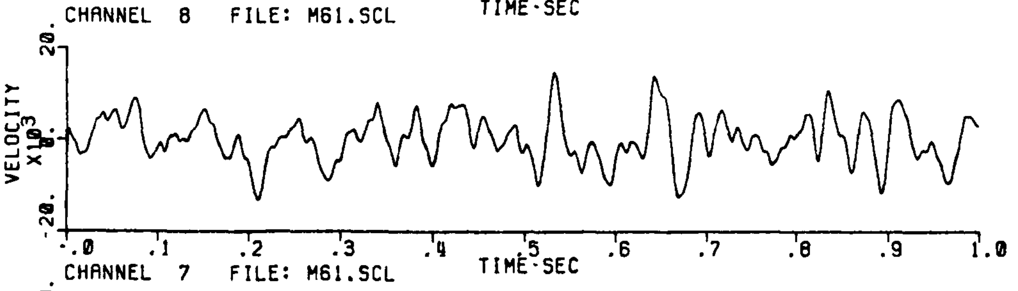
Channels 7-12 Seismic Data Direction B Range 2

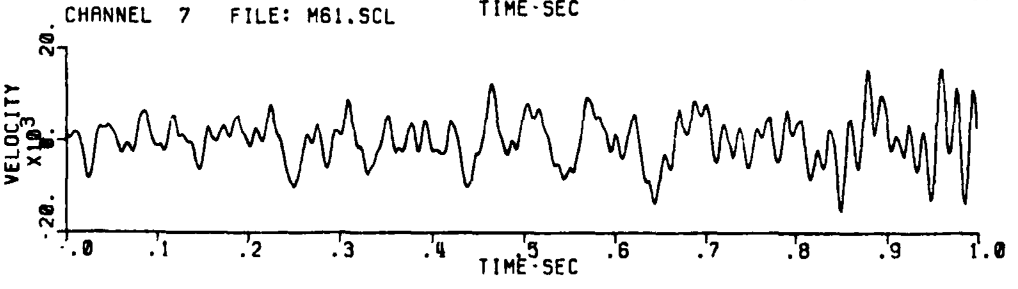

CHANNEL 12 FILE: M61.SCL
VELOCITY X10³
TIME-SEC


CHANNEL 11 FILE: M61.SCL
VELOCITY X10³
TIME-SEC


CHANNEL 10 FILE: M61.SCL
VELOCITY X10³
TIME-SEC


CHANNEL 9 FILE: M61.SCL
VELOCITY X10³
TIME-SEC


CHANNEL 8 FILE: M61.SCL
VELOCITY X10³
TIME-SEC


CHANNEL 7 FILE: M61.SCL
VELOCITY X10³
TIME-SEC